High Performance in Numerical Linear Algebra

James Demmel
EECS and Math Depts.
CITRIS Chief Scientist
UC Berkeley

5 May 2003

Outline

- National Academy Committee to Study the Future of Supercomputing (FOSC)
- Technology Trends and Methodology
- Dense Linear Algebra
- Sparse Direct Solvers for Ax= b
- Automatic Performance Tuning of Sparse Kernels
- Sparse Iterative Solvers for Ax=b

Future of Supercomputing (FOSC)

- “The committee will assess the status of supercomputing in the United States, including the characteristics of relevant systems and architecture research in government, industry, and academia and the characteristics of the relevant market. The committee will examine key elements of context--the history of supercomputing, the erosion of research investment, the needs of government agencies for supercomputing capabilities--and assess options for progress. Key historical or causal factors will be identified. The committee will examine the changing nature of problems demanding supercomputing (e.g., weapons design, molecule modeling and simulation, cryptanalysis, bioinformatics, climate modeling) and the implications for systems design. It will seek to understand the role of national security in the supercomputer market and the long-term federal interest in supercomputing. An interim report will be delivered July, 2003. The committee's work will culminate in a report of its assessment, including recommendations, which will be disseminated in relevant segments of government, industry, and the academic research community.”

FOSC Membership

- Sue Graham – UCB
  - Cochair
- Marc Snir – UIUC
  - Cochair
- William Dally – Stanford
- James Demmel – UCB
- Jack Dongarra – UTenn
- Ken Flamm – UT Austin
- Mary Jane Irwin – Penn St
- Charles Koelbel – Rice
- Butler Lampson - Microsoft
- Robert Lucas - ISI
- Paul Messina – ANL
- Jeffrey Perloff – UCB
- William Press – LANL
- Albert Semtner – Naval PGS
- Scott Stern – Northwestern
- Shankar Subramaniam – UCSD
- Lawrence Turbell – Eagle Alliance
- Steven Wallach – Chiaro Networks

www7.nationalacademies.org/cstb/project_supercomputing.html

FOSC Status

- First Meeting – March 2003
- Currently information gathering
- Interim report – July 2003
- Final report: end of 2004

Tech Trends & Methodology

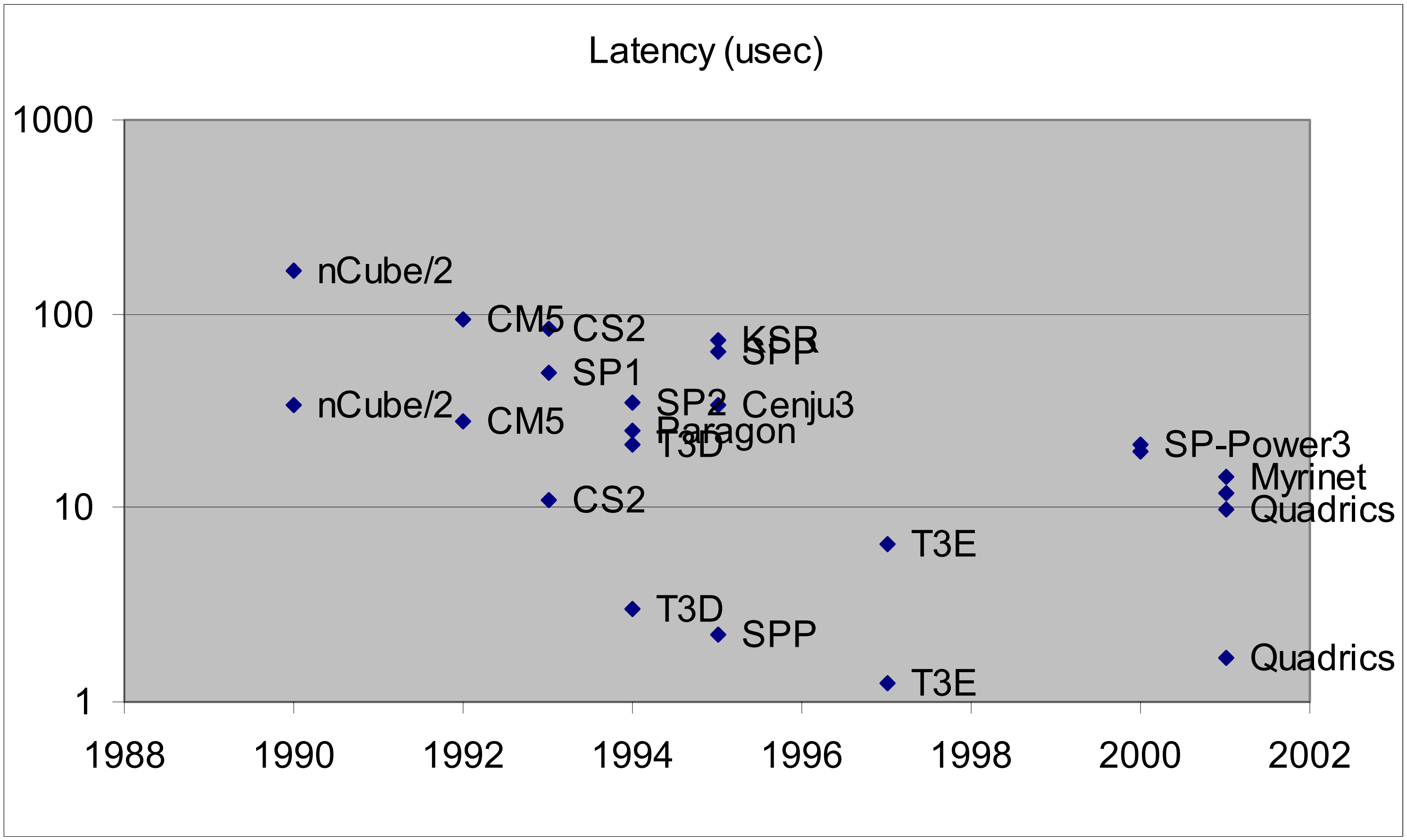
- **Performance depends on**
  - **Maximum Flop rate**
  - **Memory latencies and bandwidths**
  - **Network latencies and bandwidths**
- **The Flop rate is growing faster than the other quantities**
  - **Latencies improving most slowly**
  - **Moving to Grid makes things worse**
- **Methodology**
  - **Develop performance models**
  - **Plug tech trend lines into models**
  - **Predict scalability, identify bottlenecks**
  - **Fix bottlenecks or find new algorithms**
  - **Evaluate architectural designs …**
- **Work in progress…**

Winner of TOPS 500, by year

Year	Machine	Tflops	Factor faster	Peak Tflops	Num Procs	N
2002	**Earth System Computer, NEC**	35.6	4.9	40.8	5104	1.04M
2001	**ASCI White, IBM SP Power 3**	7.2	1.5	11.1	7424	.52M
2000	**ASCI White, IBM SP Power 3**	4.9	2.1	11.1	7424	.43M
1999	**ASCI Red, Intel PII Xeon**	2.4	1.1	3.2	9632	.36M
1998	**ASCI Blue, IBM SP 604E**	2.1	1.6	3.9	5808	.43M
1997	**ASCI Red, Intel Ppro, 200 MHz**	1.3	3.6	1.8	9152	.24M
1996	**Hitachi CP-PACS**	.37	1.3	.6	2048	.10M
1995	**Intel Paragon XP/S MP**	.28	1	.3	6768	.13M

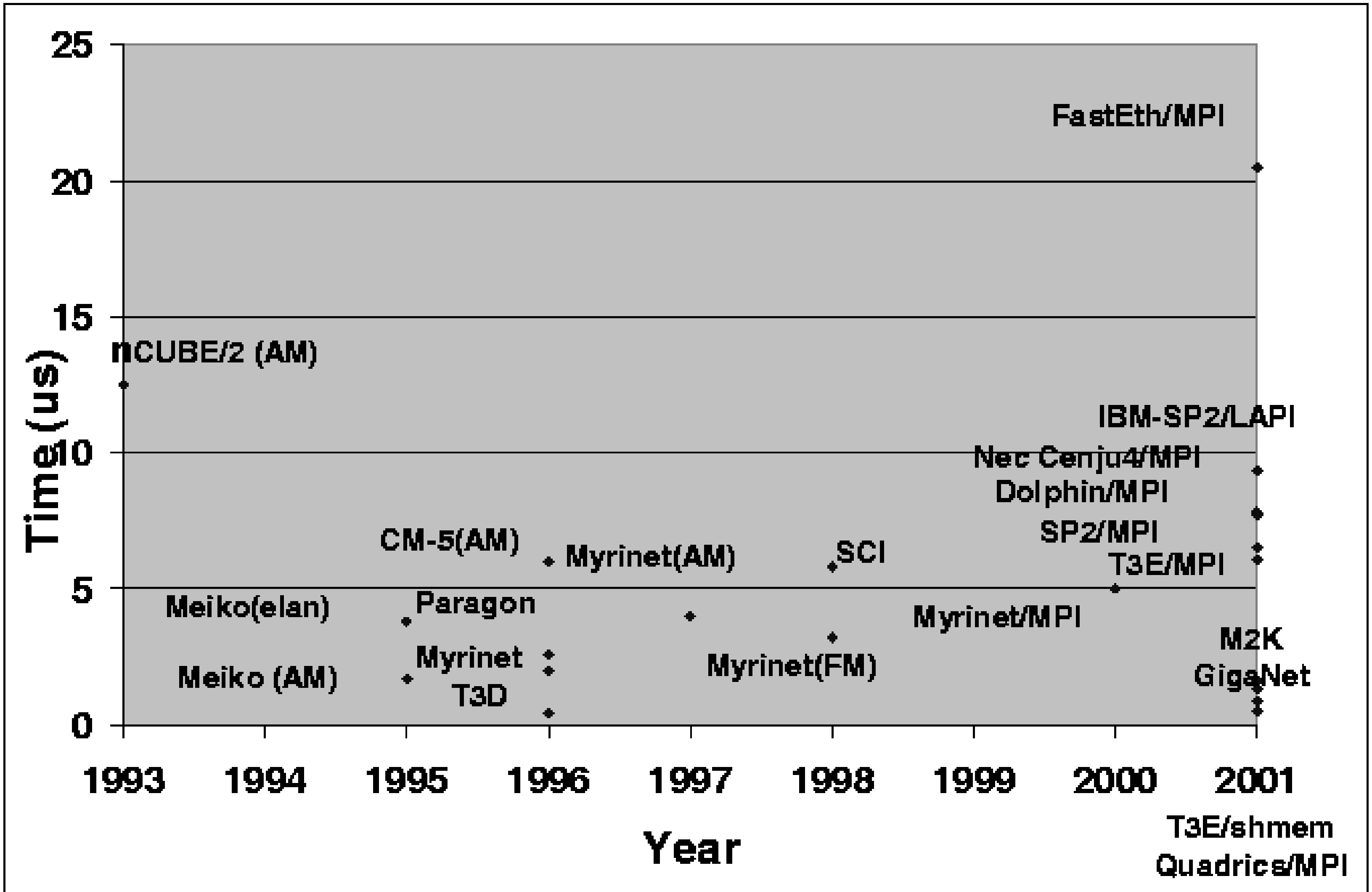
Source: Jack Dongarra (UTK)

End-to-end message latencies are not improving (much)



Source: K. Yelick (UCB), C. Iancu (NERSC)

Software Overhead is a culprit



Source: K. Yelick (UCB), C. Iancu (NERSC)

ScaLAPACK
A Parallel Distributed
Dense Linear Algebra Library

ScaLAPACK Team

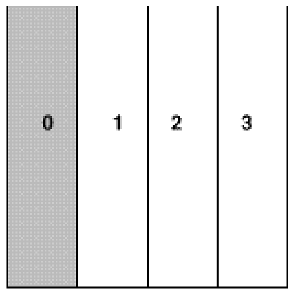
- Susan Blackford, UT
- Jaeyoung Choi, Soongsil U
- Andy Cleary, LLNL
- Ed D'Azevedo, ORNL
- Jim Demmel, UCB
- Inder Dhillon, UCB
- Christof Voemel
  CERFACS -> UCB
- Jack Dongarra, UT/ORNL
- Sven Hammarling, NAG
- Greg Henry, Intel
- Osni Marques, NERSC
- Antoine Petitet, UT
- Ken Stanley, UCB
- David Walker, Cardiff U
- Clint Whaley, UT

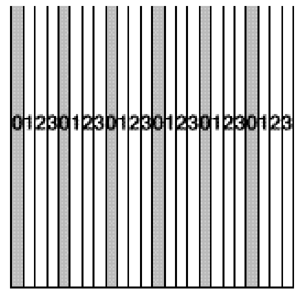
http://www.netlib.org/scalapack

scalapack@cs.utk.edu

Possible Data Layouts

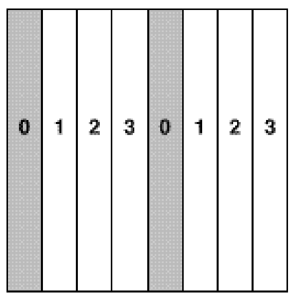
1D blocked

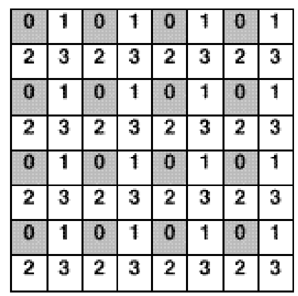




1D cyclic

1D block cyclic





2D block cyclic

- ScaLAPACK supports all layouts
- 2D block cyclic recommended, for load balance and BLAS3

Parallelism in ScaLAPACK

- Level 3 BLAS block operations
  - All the reduction routines
- Pipelining
  - QR Iteration, Triangular Solvers, classic factorizations
- Redundant computations
  - Condition estimators
- Static work assignment
  - Bisection
- Task parallelism
  - Sign function eigenvalue computations
- Divide and Conquer
  - Tridiagonal and band solvers, symmetric eigenvalue problem and Sign function
- Cyclic reduction
  - Reduced system in the band solver

ScaLAPACK Performance Models (1)

ScaLAPACK Operation Counts

$$T(N,P) = \frac{C_f N^3}{P} t_f + \frac{C_v N^2}{\sqrt{P}} t_v + \frac{C_m N}{NB} t_m, \qquad T_{seq}(N,P) = C_f N^3 t_f.$$

$$E(N,P) = \left(1 + \frac{1}{NB} \frac{C_m t_m}{C_f t_f} \frac{P}{N^2} + \frac{C_v t_v}{C_f t_f} \frac{\sqrt{P}}{N}\right)^{-1}.$$

Driver	Options	C_f	C_v	C_m
PxGESV	1 right hand side	$2/3$	$3 + 1/4 \log_2 P$	$NB\,(6 + \log_2 P)$
PxPOSV	1 right hand side	$1/3$	$2 + 1/2 \log_2 P$	$4 + \log_2 P$
PxGELS	1 right hand side	$4/3$	$3 + \log_2 P$	$2\,(NB \log_2 P + 1)$
PxSYEVX	eigenvalues only	$4/3$	$5/2 \log_2 P$	$17/2\,NB + 2$
PxSYEVX	eigenvalues and eigenvectors	$10/3$	$5 \log_2 P$	$17/2\,NB + 2$
PxSYEV	eigenvalues only	$4/3$	$5/2 \log_2 P$	$17/2\,NB + 2$
PxSYEV	eigenvalues and eigenvectors	$22/3$	$5 \log_2 P$	$17/2\,NB + 2$
PxGESVD	singular values only	$26/3$	$10 \log_2 P$	$17NB$
PxGESVD	singular values and left and right singular vectors	$38/3$	$14 \log_2 P$	$17NB$
PxLAHQR	eigenvalues only	5	$9/2\,(\sqrt{P}) * \log_2 P$ $+8\,N/NB$	$9\,(2 + \log_2 P)\,N$
PxLAHQR	full Schur form	18	$9/2\,(\sqrt{P}) * \log_2 P$ $+8\,N/NB$	$9\,(2 + \log_2 P)\,N$

ScaLAPACK Performance Models (2)
Compare Predictions and Measurements

IBM SP2[a]	P	Values of N									
		2000		5000		7500		10000		15000	
		Est	Obt	Est	Obt	Est	Obt	Est	Obt	Est	Obt
PDGESV (LU)	4	357	421	632	603						
	16	497	722	1581	1543	2116	1903	2424	2149		
	64	502	924	2432	3017	4235	4295	5793	5596	7992	7057
PDPOSV (Cholesky)	4	530	462	669	615						
	16	1315	1081	2083	1811	2366	2118	2535	2312		
	64	2577	1807	5327	4431	6709	5727	7661	6826	8887	8084

[a] One process spawned per node and one computational IBM POWER2 590 processor per node.

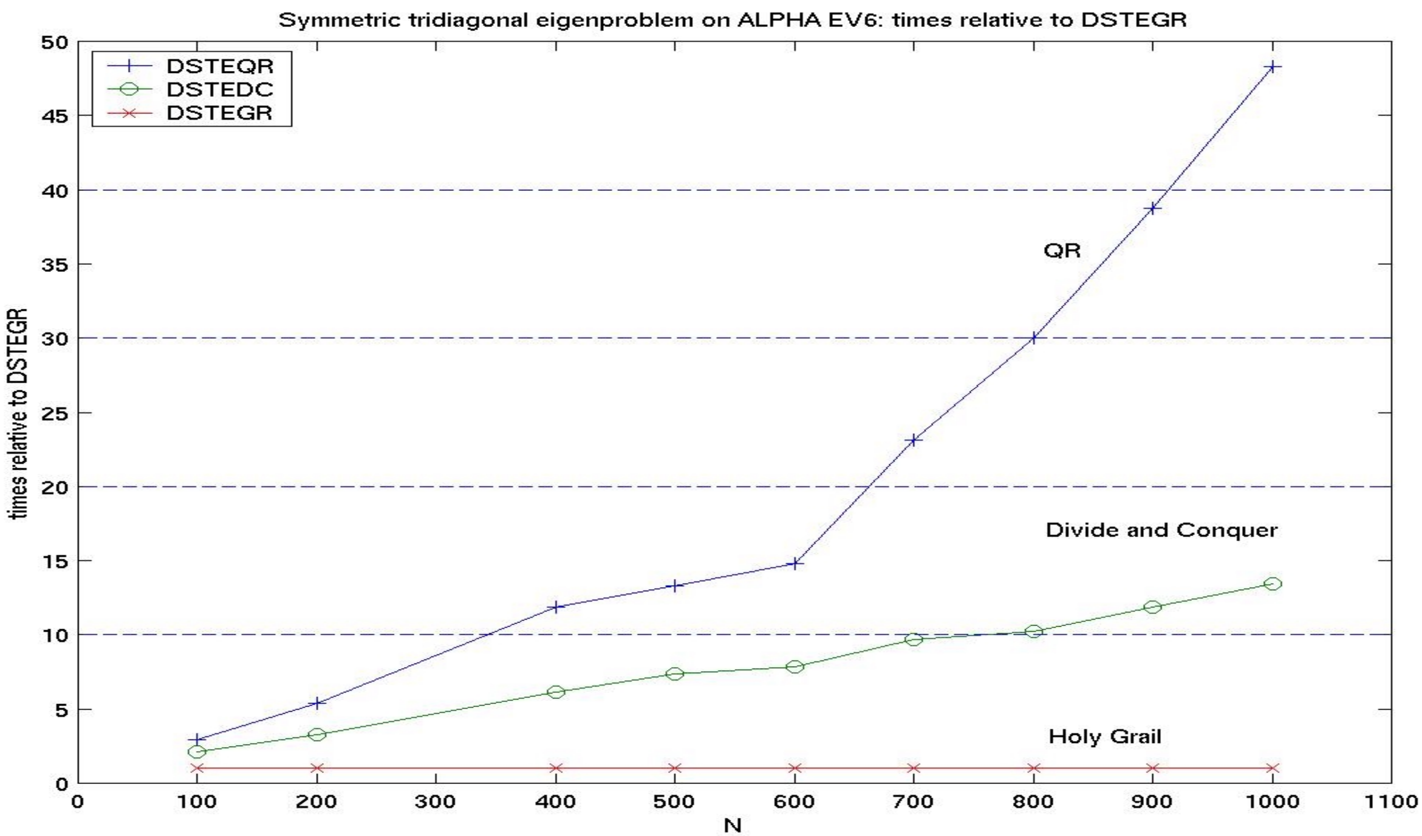
Making the nonsymmetric eigenproblem scalable

- $Ax_i = \lambda_i x_i$, Schur form $A = QTQ^T$
- Parallel HQR
  - Henry, Watkins, Dongarra, Van de Geijn
  - Now in ScaLAPACK
  - Not as scalable as LU: N times as many messages
  - Block-Hankel data layout better in theory, but not in ScaLAPACK
- Sign Function
  - Beavers, Denman, Lin, Zmijewski, Bai, Demmel, Gu, Godunov, Bulgakov, Malyshev
  - $A_{i+1} = (A_i + A_i^{-1})/2 \rightarrow$ shifted projector onto Re $\lambda > 0$
  - Repeat on transformed A to divide-and-conquer spectrum
  - Only uses inversion, so scalable
  - Inverse free version exists (uses QRD)
  - Very high flop count compared to HQR, less stable

Making the symmetric eigenproblem and SVD scalable

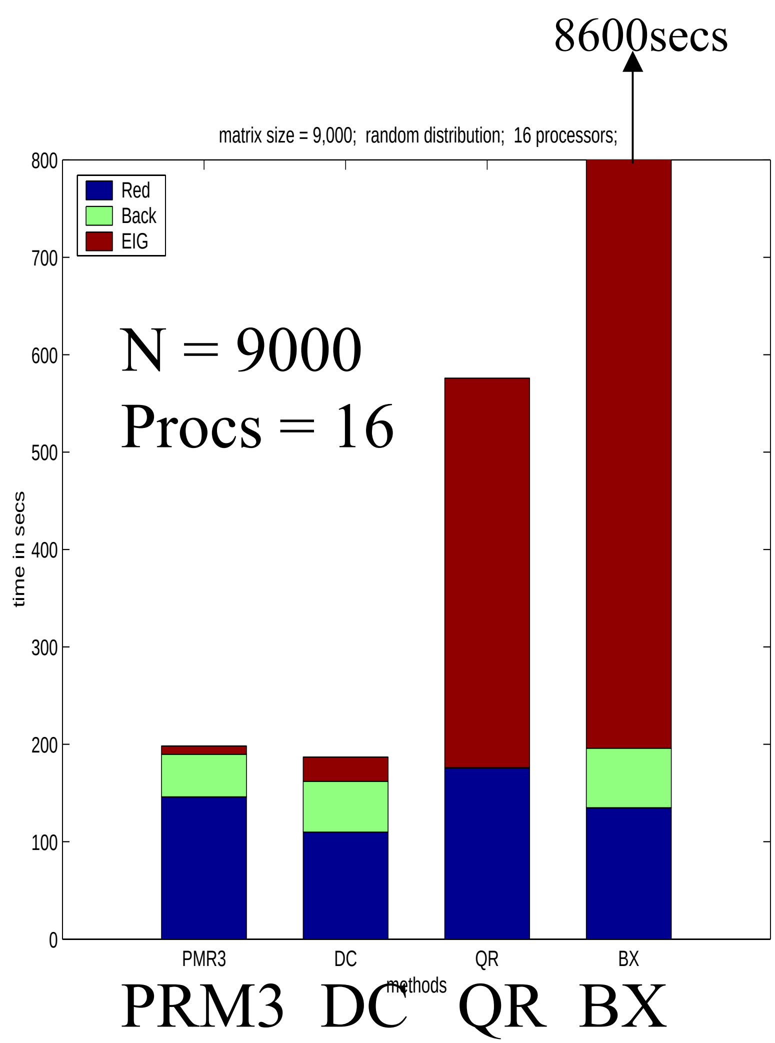
The "Holy Grail" (Parlett, Dhillon, Marques)

Perfect Output complexity (O(n * #vectors)), Embarrassingly parallel, Accurate



To be propagated throughout LAPACK and ScaLAPACK

Holy Grail Preliminary Results



- Dhillon, van de Geijn, Bientinesi
- Buffalo CCR Linux Cluster
  - 300 dual P4 nodes + Myricom
- PRM3 = PLAPACK Holy Grail
- DC = ScaLAPACK D&C
- QR = PLAPACK QR
- BX = ScaLAPACK Bis+Invit
- Best PRM3 results so far
  - 64 procs, n = 64K
  - Reduction = 11K secs
  - Evals = 390 secs
  - Evecs = 50 secs
  - Backtrans = 3K secs

ScaLAPACK
Summary and Conclusions

- "One-sided Problems" are scalable
  - LU ("Linpack Benchmark")
  - Cholesky, QR
- "Two-sided Problems" are harder
  - At least half BLAS2, not all BLAS3
    - Better reduction routines for SVD (Howell & Fulton, Lang & Grosser)
  - Eigenproblems, SVD (Holy Grail coming…)
    - Christof Voemel to join team as soon as visa processed…
- Narrow band problems hardest
  - Solving and eigenproblems
  - Galois theory of parallel prefix
- www.netlib.org/scalapack

Parallel Distributed Sparse Gaussian Elimination

Xiaoye Li
Laura Grigori
Jason Riedy

Phases of Sparse Direct Solvers

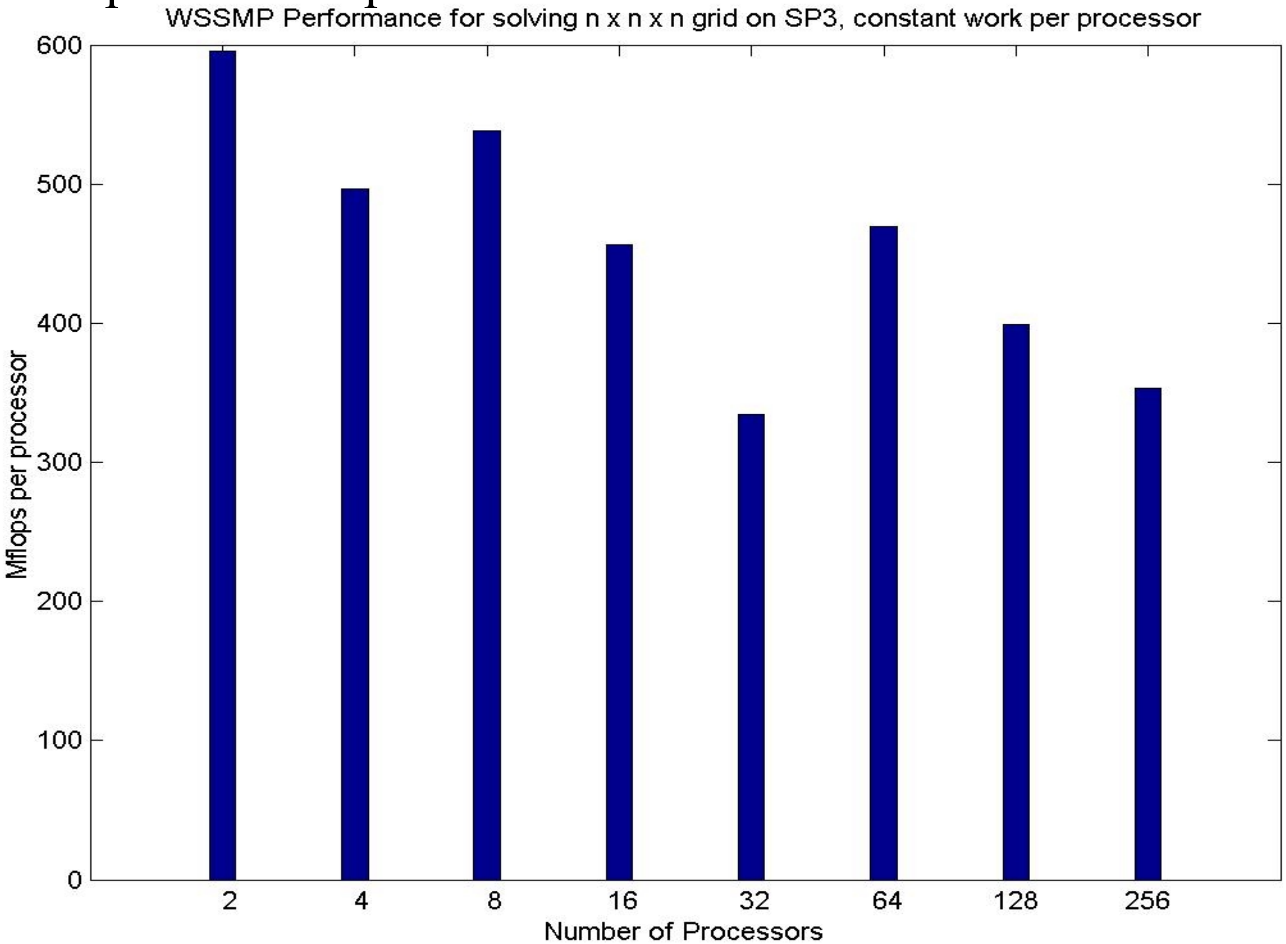
- Ordering
  - Choose P_r and P_c, Set $A' = P_r*A*P_c^T$
  - Maximize sparsity, parallelism
  - NP-hard, so must approximate
- Symbolic factorization
  - Compute data structures for L and U where $A'=L*U$
- Numeric factorization
  - Compute L and U
  - May need to further permute or modify A' (pivoting)
  - Usually the bottleneck
- Triangular solve
  - Solve $A'x' = LUx' = b'$ by substitution, x' and b' permuted

"Easy case": When $A = A^T > 0$

- Cholesky, stable for any $P_r = P_c$
  - Can choose P_r just for sparsity and parallelism
- All phases can be parallelized
- PSPASES and WSSMP
  - Joshi, Karypis, Kumar, Gupta, Gustavson
  - Sub (elimination) tree to sub-cube mapping
- Performance model 1
  - Matrix from 5 pt Laplacian on n x n (2D) mesh, Nested dissection
  - $N = n^2$
  - Parallel time = $\mathbf{O(t_f N^{3/2} / P + t_v (N / P^{1/2} + N^{1/2} + P \log P))}$
- Performance model 2
  - Matrix from 11 pt Laplacian on n x n x n (3D) mesh, Nested dissection
  - $N = n^3$
  - Parallel time = $\mathbf{O(t_f N^2 / P + t_v (N^{4/3} / P^{1/2} + N^{2/3} + P \log P))}$

Scalability of WSSMP on SP3 for n x n x n mesh

- 128 node SP3 with 2-way SMP 200 MHz Power3 nodes
- Scale $N^2 = n^6$ with P for constant work per processor
- Performance model 2 says efficiency should drop – it does
- Up to 92 Gflops

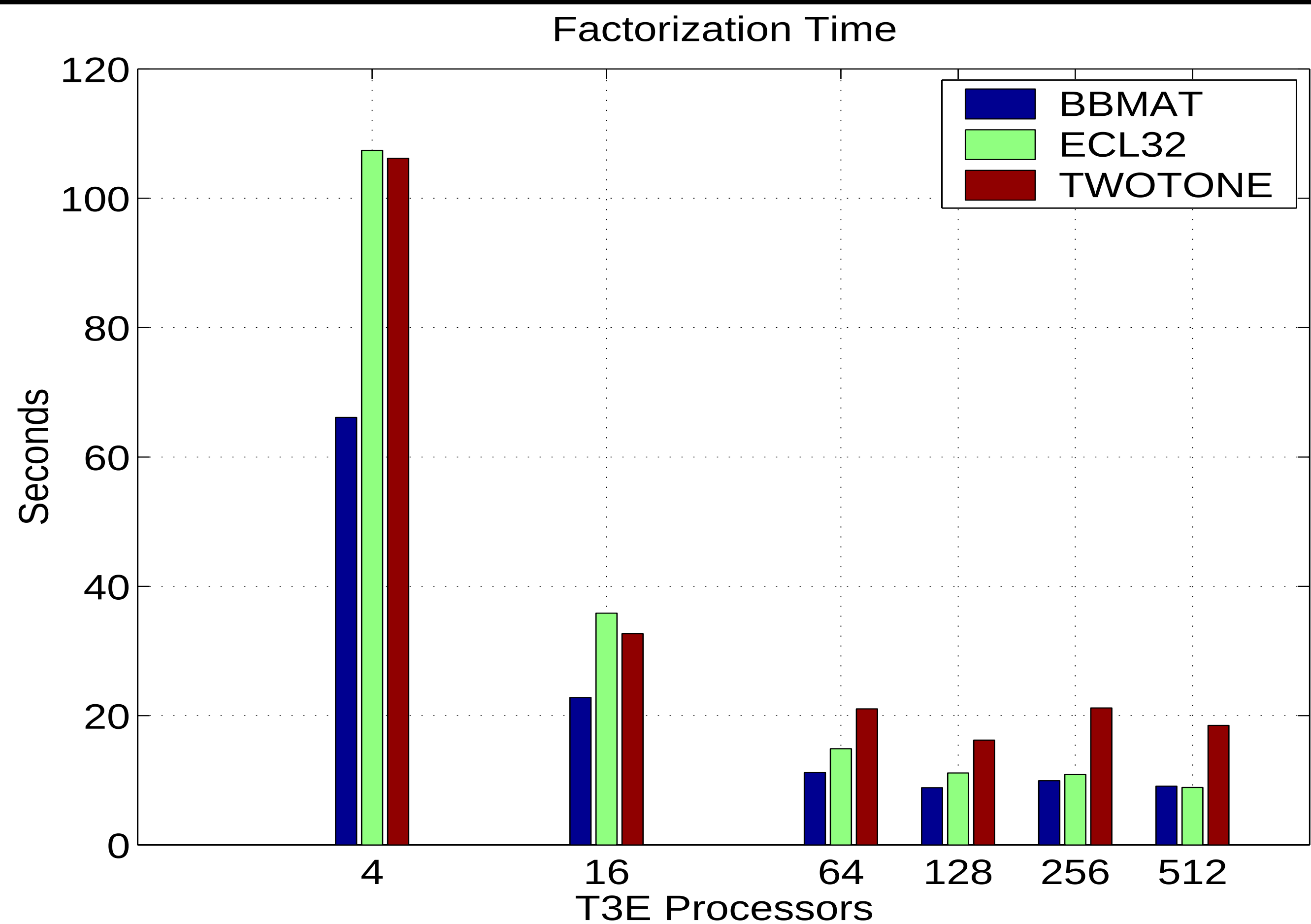


Hard case: General A

- Arbitrary P_r , P_c may lose stability
  - Usual partial pivoting solution has too many small messages
  - Can we still scale as well as Cholesky?
- MUMPS (Amestoy, Duff, L'Excellent)
  - Multifrontal, threshold pivoting
  - Parallelism from E-tree and 2D blocking of root
  - Permute, scale A to maximize diagonal: $D_r PAD_c = A'$
    - Reduces fill, deferred pivots
- SuperLU (Li, Demmel)
  - Right looking, static pivoting + iterative refinement
    - Static pivoting => similar techniques as Cholesky available
  - Permute and scale A as above
    - critical for stability
  - Replace tiny pivots by $\sqrt{\varepsilon}\,||A||$
  - Parallelism from 2D block cyclic layout
- Only numeric phases are parallel so far

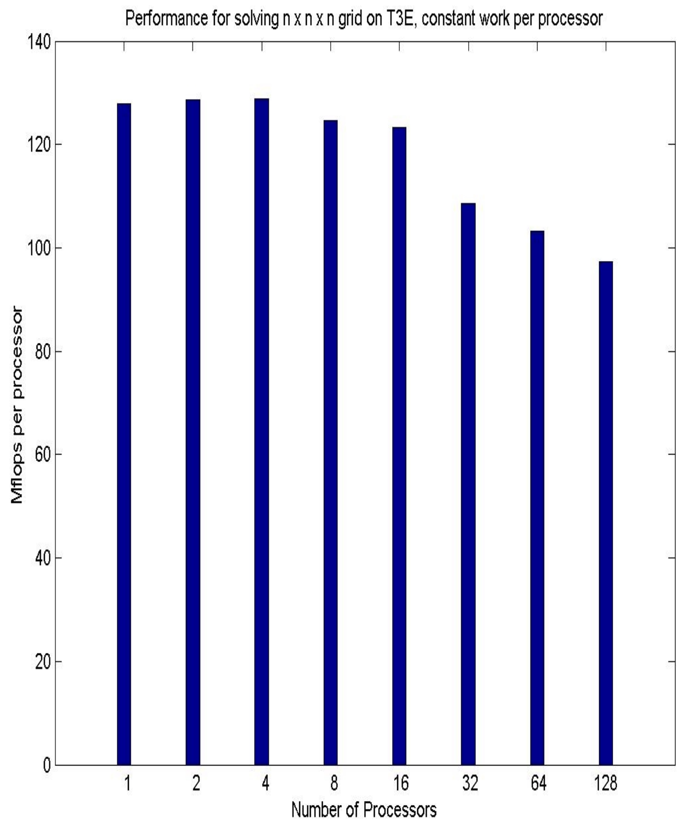
SuperLU Examples

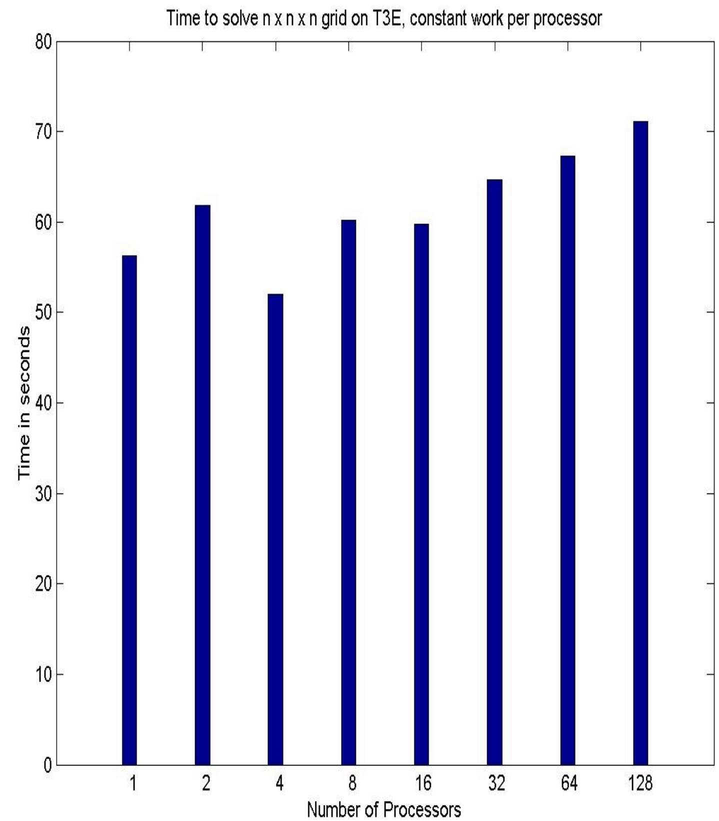
Matrix	Source	Symm	N	Nnz(A)	Nnz(L+U)	GFlops
BBMAT	Fluid flow	.54	38,744	1.77M	40.2M	31.2
ECL32	Device sim.	.93	51,993	.38M	42.7M	68.4
TWOTONE	Circuit sim.	.43	120,750	1.22M	11.9M	8.0



Scalability of SuperLU on n x n x n Mesh

- T3E: Scale $N^2 = n^6$ with P for constant work per processor
- Up to 12.5 Gflops on 128 procs
- Similar scalability to Cholesky on same problems





- SP3: n = 100, N = 1M, 49 Gflops (267 secs)

Adoptions of SuperLU
(Shameless Advertising)

- Industrial
  - HP (distributed, imminent)
  - Matlab, Sun, Boeing (sequential, planned)
  - NAG (sequential and multithreaded, planned)
  - FEMLAB (sequential, done)
  - Python (sequential, done)
- Academic/Lab
  - Omega3P (SLAC, accelerator design, imminent)
  - Trilinos (Sandia, imminent)
  - OpenSees (UCB, NPACI, earthquake simulation, done)
  - Dspice (Sandia, circuit simulation, done)
  - NIKE (LLNL, done)
  - PETSc (ANL, done)
  - Hypre (LLNL, done)

Future Work on Sparse Direct Solvers

- Xiaoye Li, Laura Grigori, Jason Riedy
- Parallelize all phases
  - Choice of static pivots
  - Symbolic factorization
  - Triangular solves
  - More choices of iterative clean-up schemes
    - GMRES, QMR, …
    - Extra precision
- Incomplete LU

Sparse Direct Solvers Summary and Conclusions

- Good implementations of Cholesky and LU
- Can be as scalable as dense case
  - Dense isoefficiency: $p = c\ N^2$
  - 3D cube isoefficiency: $p = c\ N^{4/3}$
  - 2D cube isoefficiency: $p = c\ N$
  - In all cases, isoefficiency if work = $c' p^{3/2}$
  - In all cases, isoefficiency if space/proc = c'' or $c'' \log p$
- More sensitive to latency
- Need more families of large unsymmetric test matrices
- www.nersc.gov/~xiaoye
- "Eigentemplates" www.netlig.org/etemplates for survey

Automatic Performance Tuning of Numerical Kernels

BeBOP: Berkeley Benchmarking and Optimization

bebop.cs.berkeley.edu

Performance Tuning Participants

- **Faculty**
  - **Jim Demmel, Kathy Yelick**
- **Researchers**
  - **David Bailey (LBL), Parry Husbands (LBL), Xiaoye Li (LBL), Lenny Oliker (LBL)**
- **PhD Students**
  - **Rich Vuduc, Yozo Hida, Geoff Pike**
- **Undergrads**
  - **Brian Gaeke , Jen Hsu, Shoaib Kamil, Suh Kang, Hyun Kim, Gina Lee, Jaeseop Lee, Michael de Lorimier, Jin Moon, Randy Shoopman, Brandon Thompson, Rajesh Nishtala, Chris Hsu, Ben Lee**

Conventional Performance Tuning

- Motivation: performance of many applications dominated by a few kernels
- Vendor or user hand tunes kernels
- Drawbacks:
  - Time consuming, tedious
  - Growing list of kernels to tune
    - Example: New BLAS Standard
  - Hard to predict performance even with intimate knowledge compiler, architecture knowledge
    - Must be redone for new architectures and compilers
    - Compiler technology often lags architecture
  - Not just a compiler problem:
    - Best algorithm may depend on input, so some tuning at run-time.
    - Not all algorithms semantically or mathematically equivalent

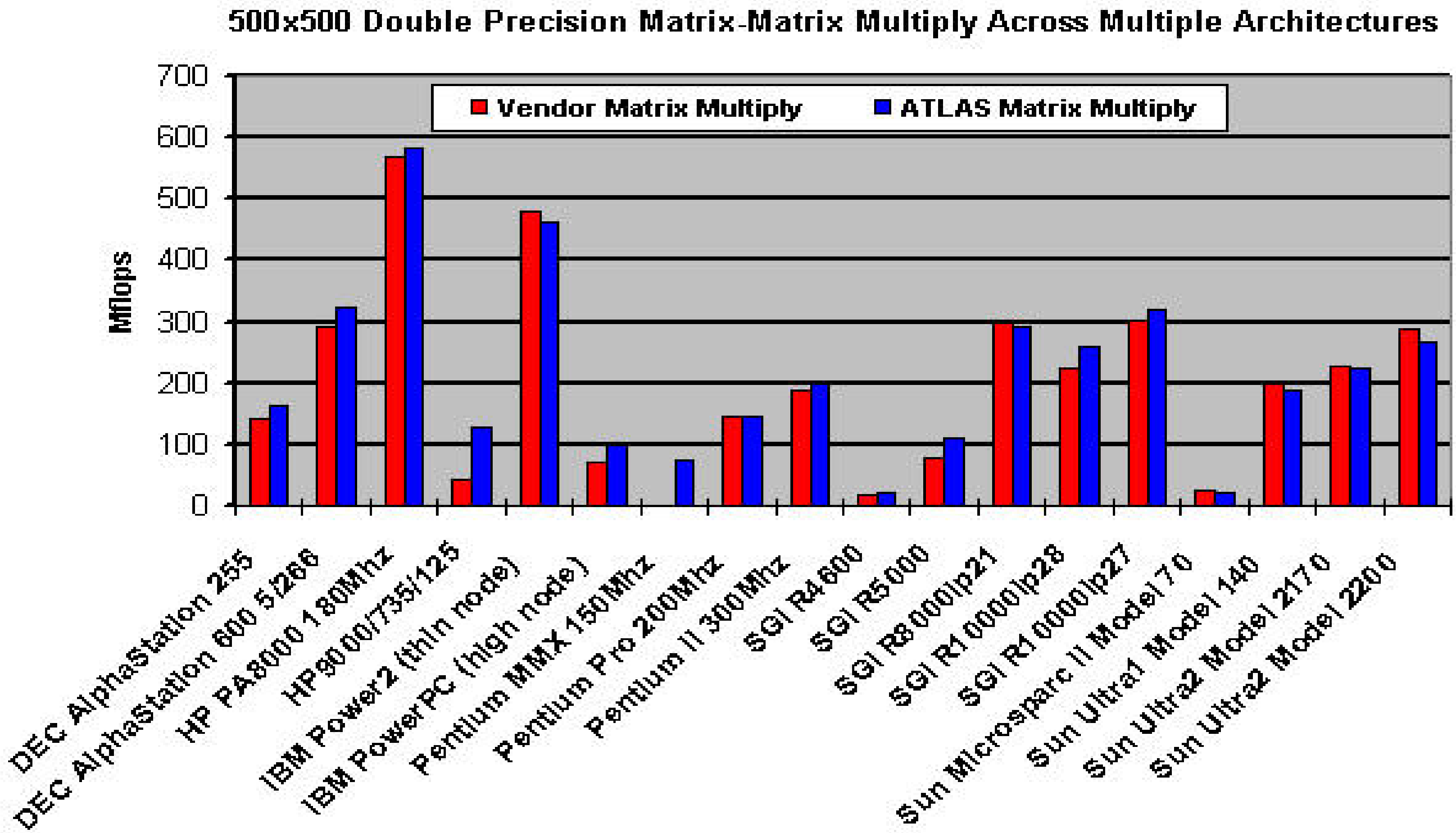
Automatic Performance Tuning

- Approach: for each kernel
  1. Identify and generate a space of algorithms
  2. Search for the fastest one, by running them
- What is a space of algorithms?
  - Depending on kernel and input, may vary
    - instruction mix and order
    - memory access patterns
    - data structures
    - mathematical formulation
- When do we search?
  - Once per kernel and architecture
  - At compile time
  - At run time
  - All of the above

Some Automatic Tuning Projects

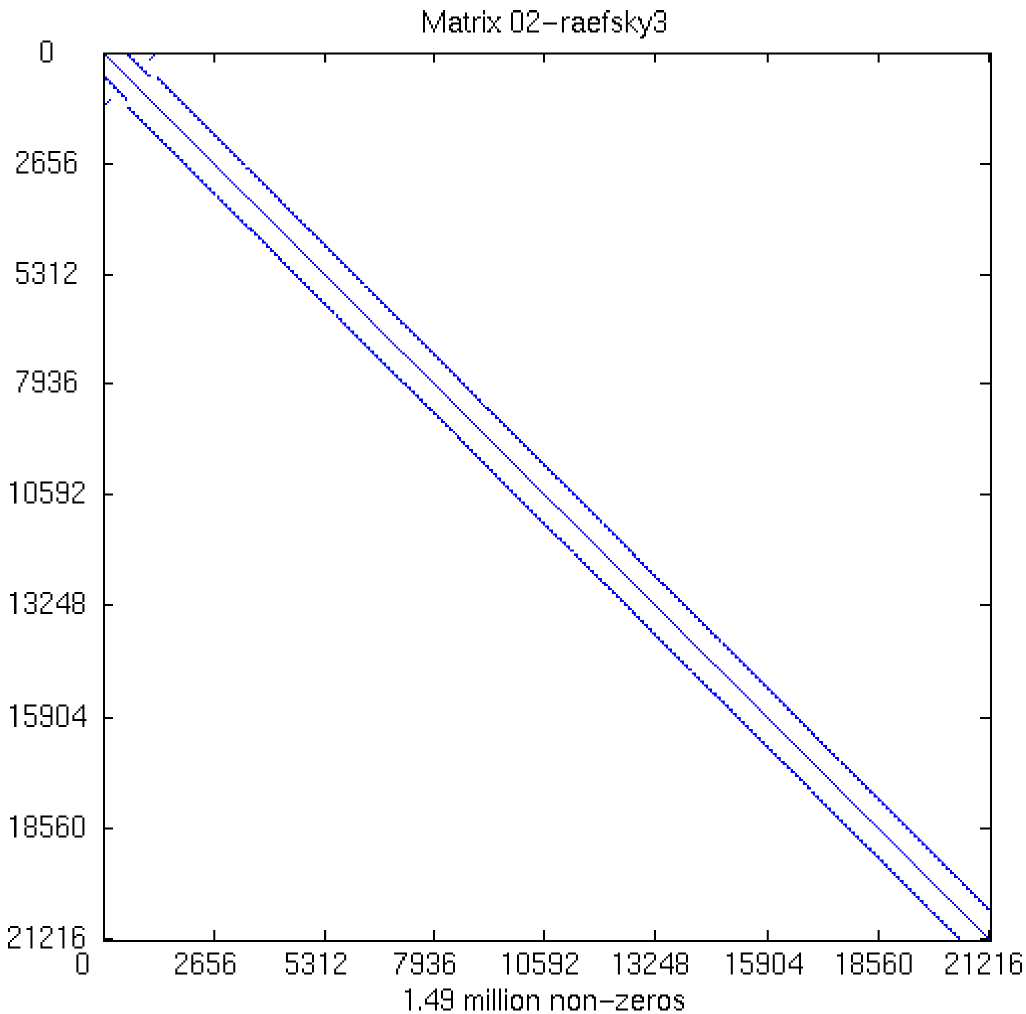
- PHIPAC (www.icsi.berkeley.edu/~bilmes/phipac) (Bilmes,Asanovic,Vuduc,Demmel)
- ATLAS (www.netlib.org/atlas) (Dongarra, Whaley; in Matlab)
- XBLAS (www.nersc.gov/~xiaoye/XBLAS) (Demmel, X. Li)
- Sparsity (www.cs.berkeley.edu/~yelick/sparsity) (Yelick, Im)
- Communication topologies (Dongarra)
- FFTs and Signal Processing
  - FFTW (www.fftw.org)
    - Won 1999 Wilkinson Prize for Numerical Software
  - SPIRAL (www.ece.cmu.edu/~spiral)
    - Extensions to other transforms, DSPs
  - UHFFT
    - Extensions to higher dimension, parallelism
- Special session at ICCS 2001
  - Organized by Yelick and Demmel
  - www.ucalgary.ca/iccs (proceedings available)
  - Pointers to other automatic tuning projects at
    - www.cs.berkeley.edu/~yelick/iccs-tune

Tuning pays off – ATLAS
(Dongarra, Whaley)



Extends applicability of PHIPAC
Incorporated in Matlab (with rest of LAPACK)

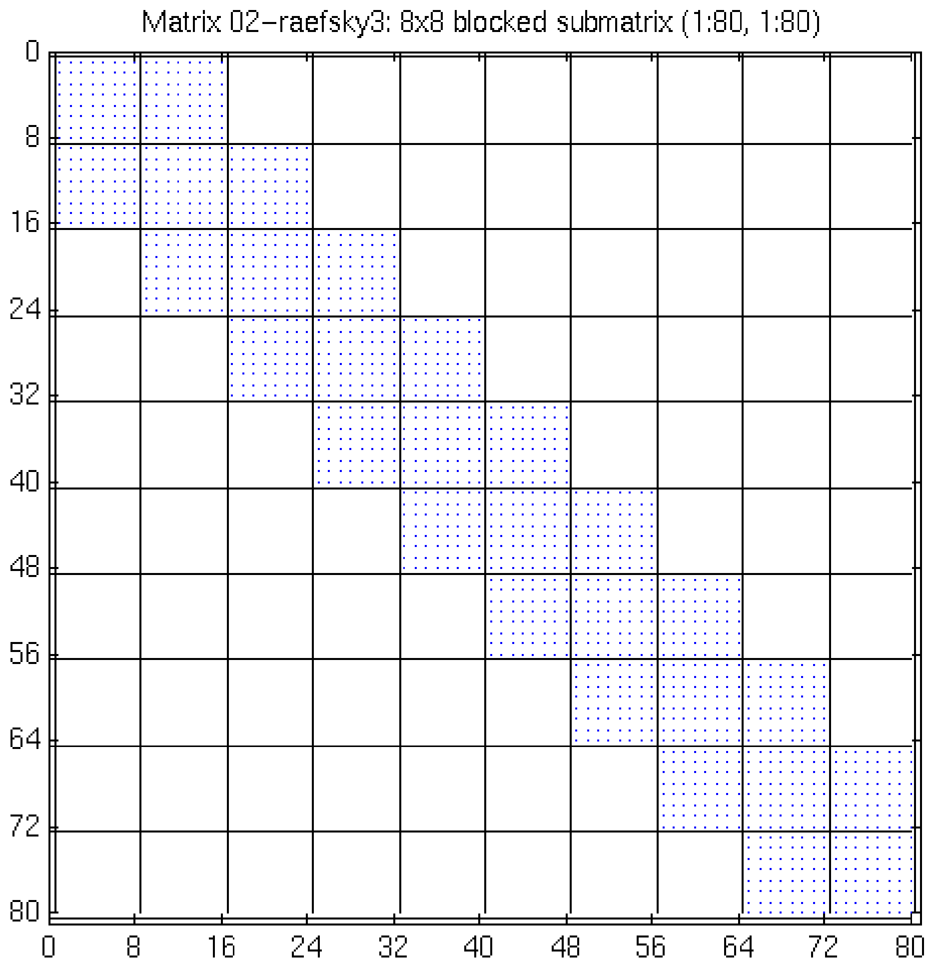
Sparse Matrix Example



- n = 21216
- nnz = 1.5 M
- kernel: SpMV

- Source: NASA structural analysis problem

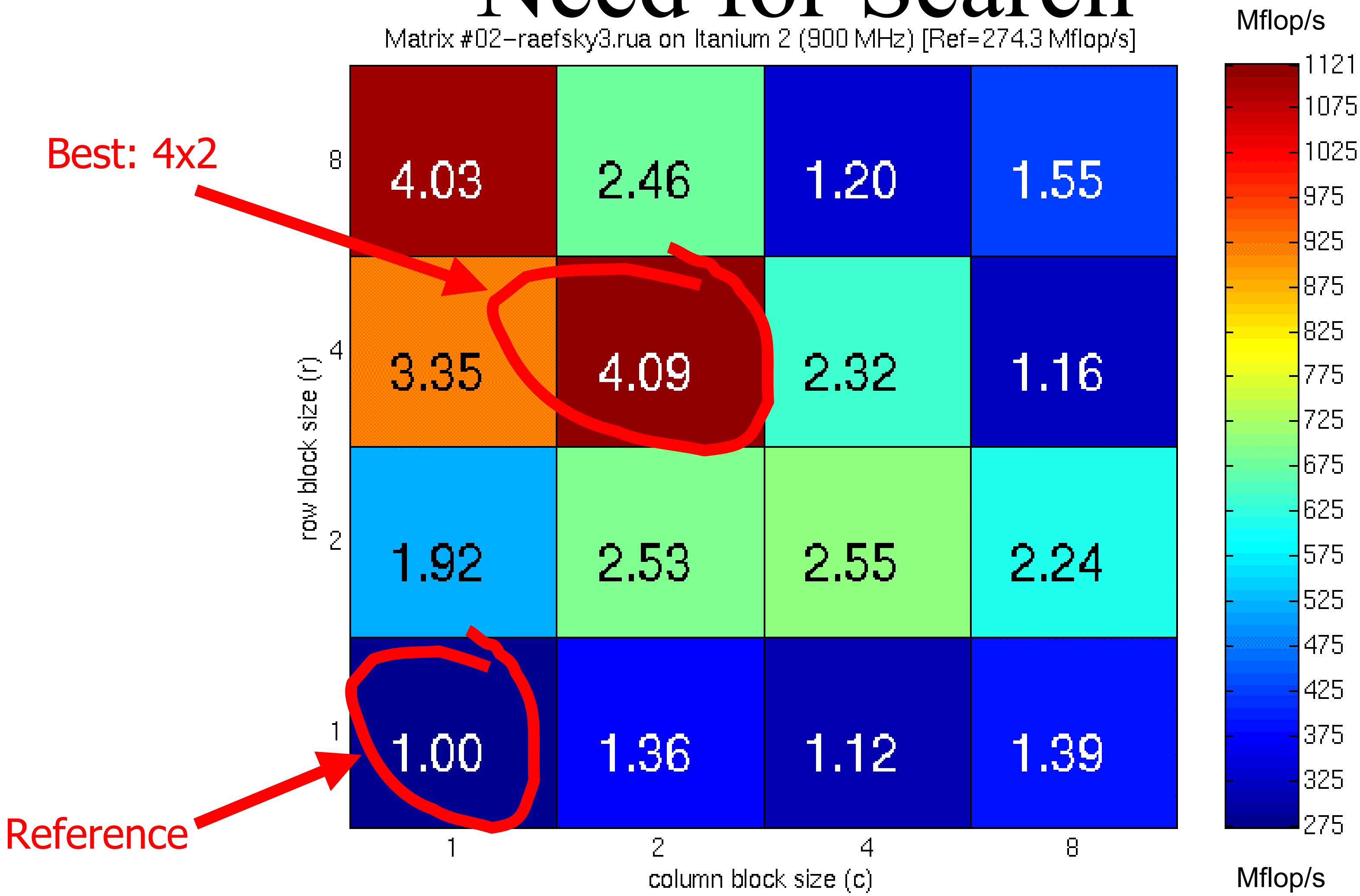
Sparse Matrix Example (enlarged submatrix)



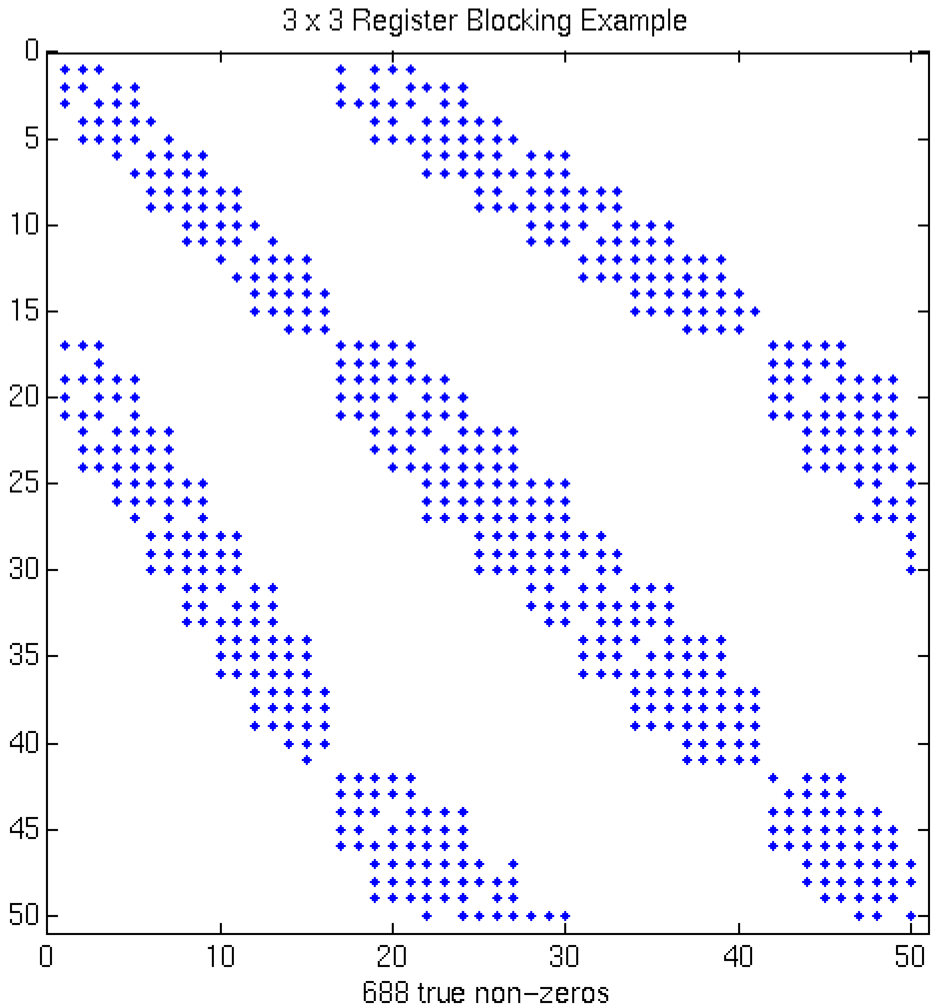
- n = 21216
- nnz = 1.5 M
- kernel: SpMV

- Natural 8x8 dense block structure

Speedups on Itanium 2: The Need for Search

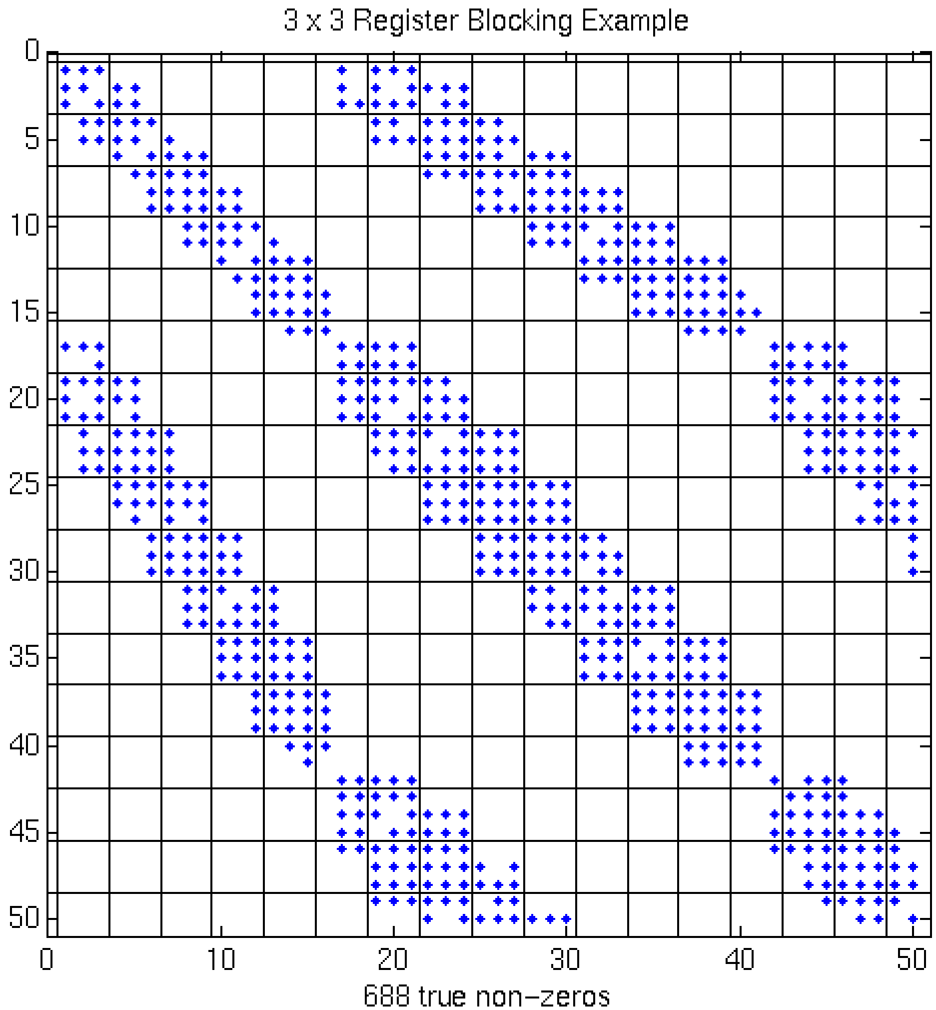


Filling-In Zeros to Improve Efficiency



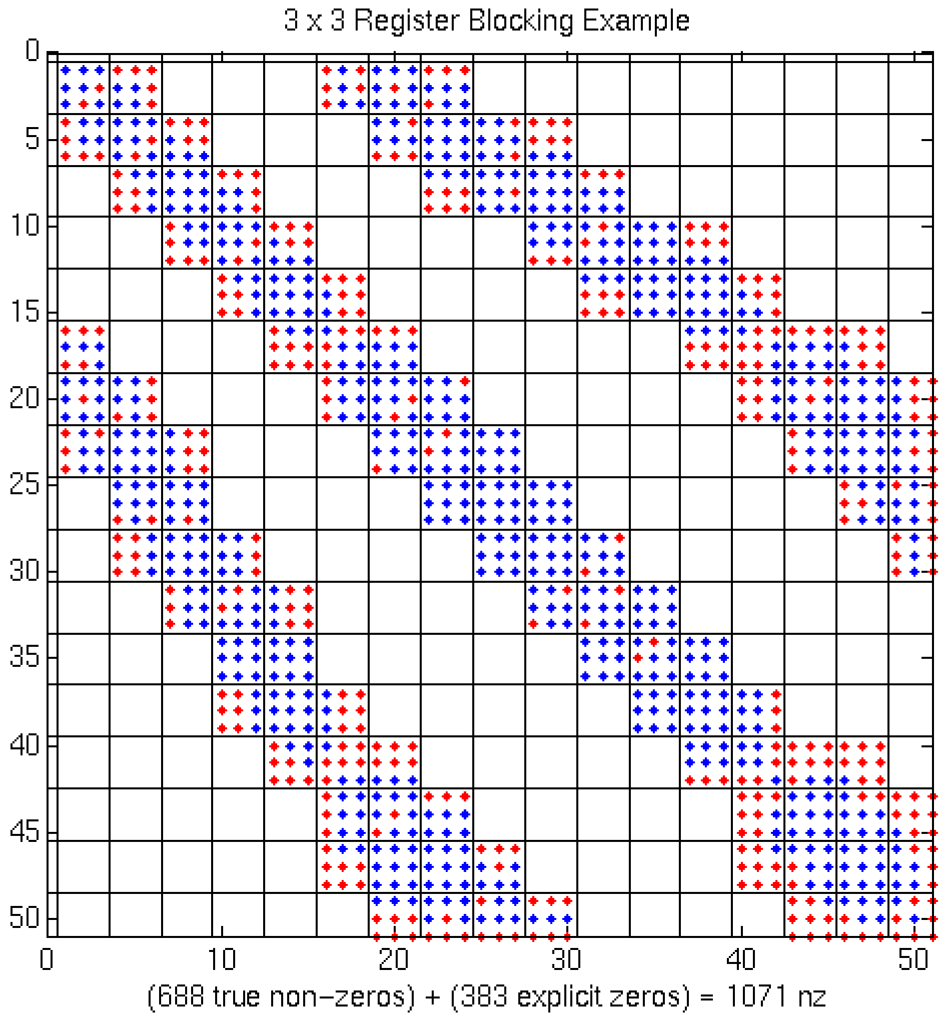
- More complicated non-zero structure in general

Filling-In Zeros to Improve Efficiency



- More complicated non-zero structure in general
- One SPARSITY technique: uniform register-level blocking
- Example: 3x3 blocking
  - Logical 3x3 grid

Filling-In Zeros to Improve Efficiency



- More complicated non-zero structure in general
- One SPARSITY technique: uniform register-level blocking

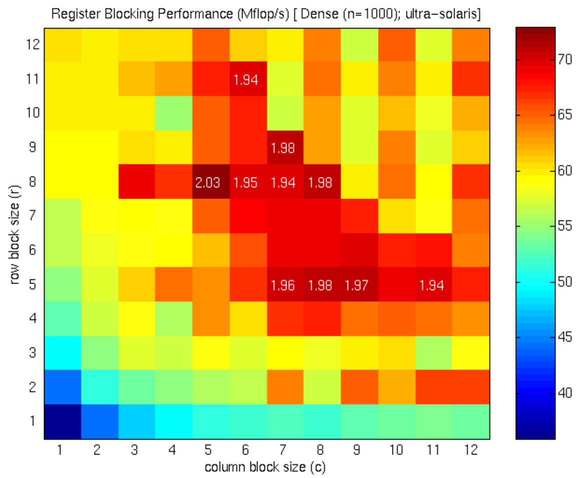
- Example: 3x3 blocking
  - Logical 3x3 grid
  - Fill-in explicit zeros
  - "Fill ratio" = 1.5
- On Pentium III: 1.5x speedup!

Tuning SpMV by Register Blocking

- Store matrix as dense r x c blocks
- Precompute performance in **Mflops** of dense A*x for various register block sizes r x c
- Given A, sample it to estimate **Fill** if A blocked for varying r x c
- Choose r x c to minimize estimated running time **Fill/Mflops**
  - Store explicit zeros in dense r x c blocks, unroll

Register-Blocked Performance of SPMV on Dense Matrices (up to 12x12)

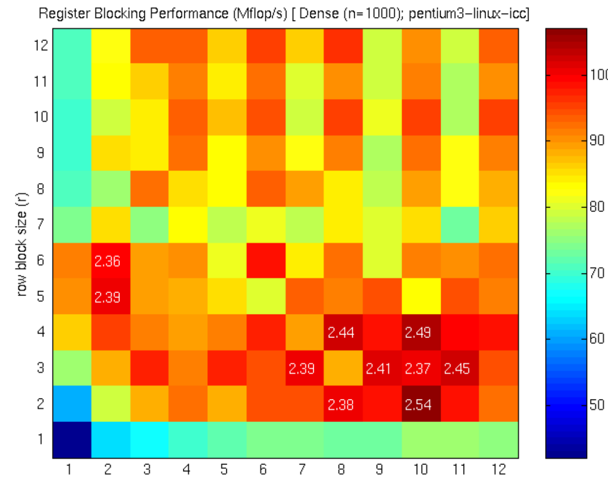
333 MHz Sun Ultra IIi



70 Mflops

35 Mflops

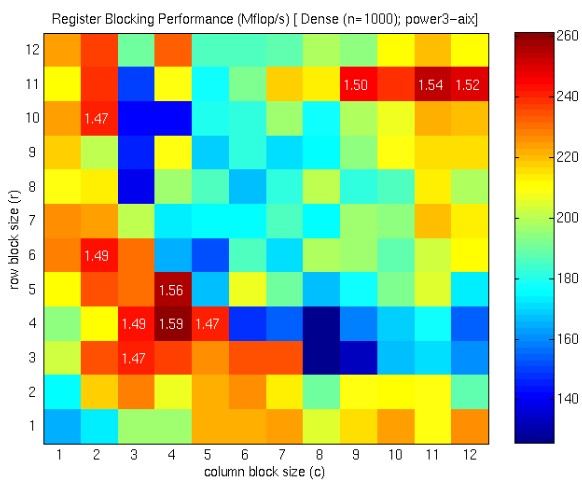
500 MHz Pentium III



110 Mflops

55 Mflops

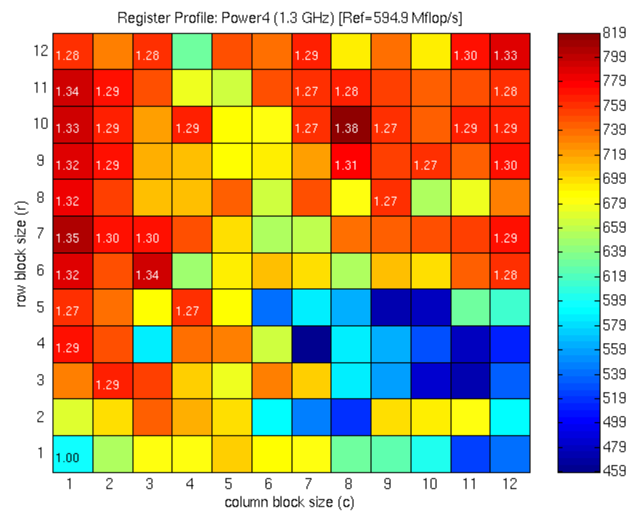
375 MHz IBM Power 3



260 Mflops

130 Mflops

1.3 GHz IBM Power 4

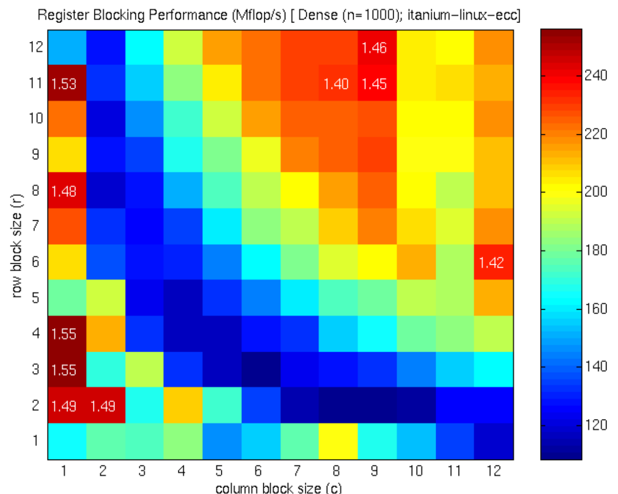


820 Mflops

460 Mflops

Register-Blocked Performance of SPMV on Dense Matrices (up to 12x12)

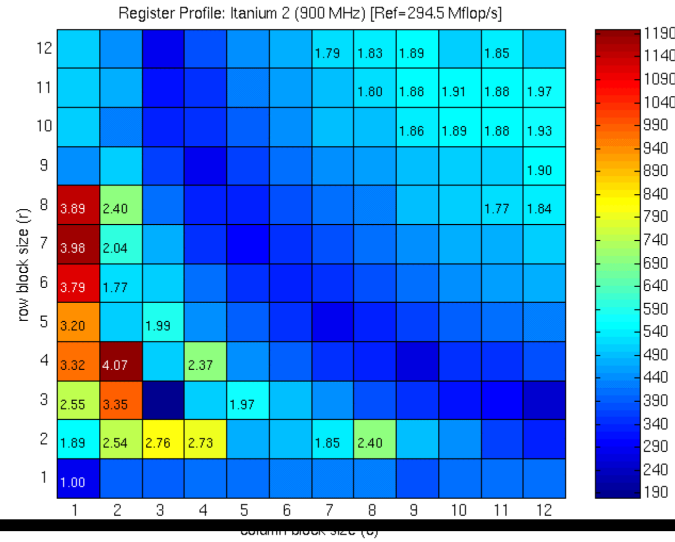
800 MHz Itanium 1



250 Mflops

110 Mflops

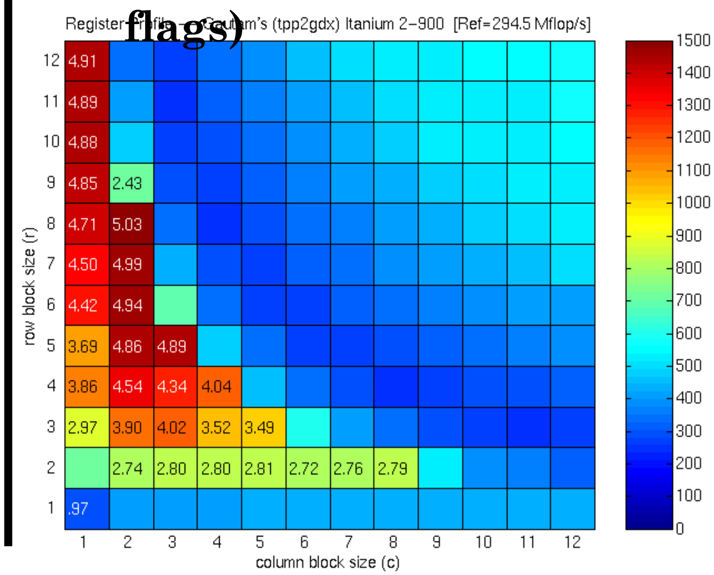
900 MHz Itanium 2



1190 Mflops

190 Mflops

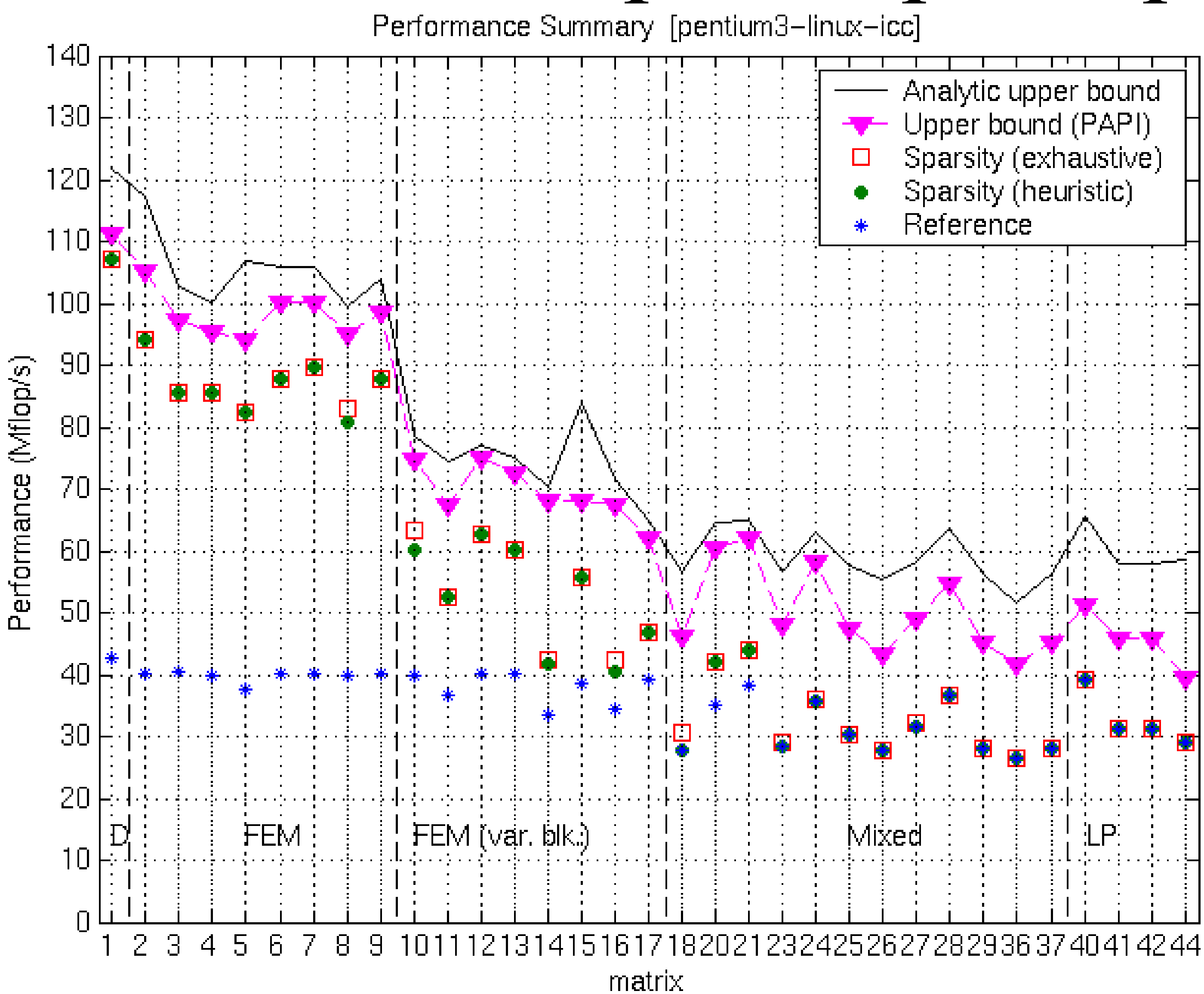
900 MHz Itanium 2 (Intel compiler flags)



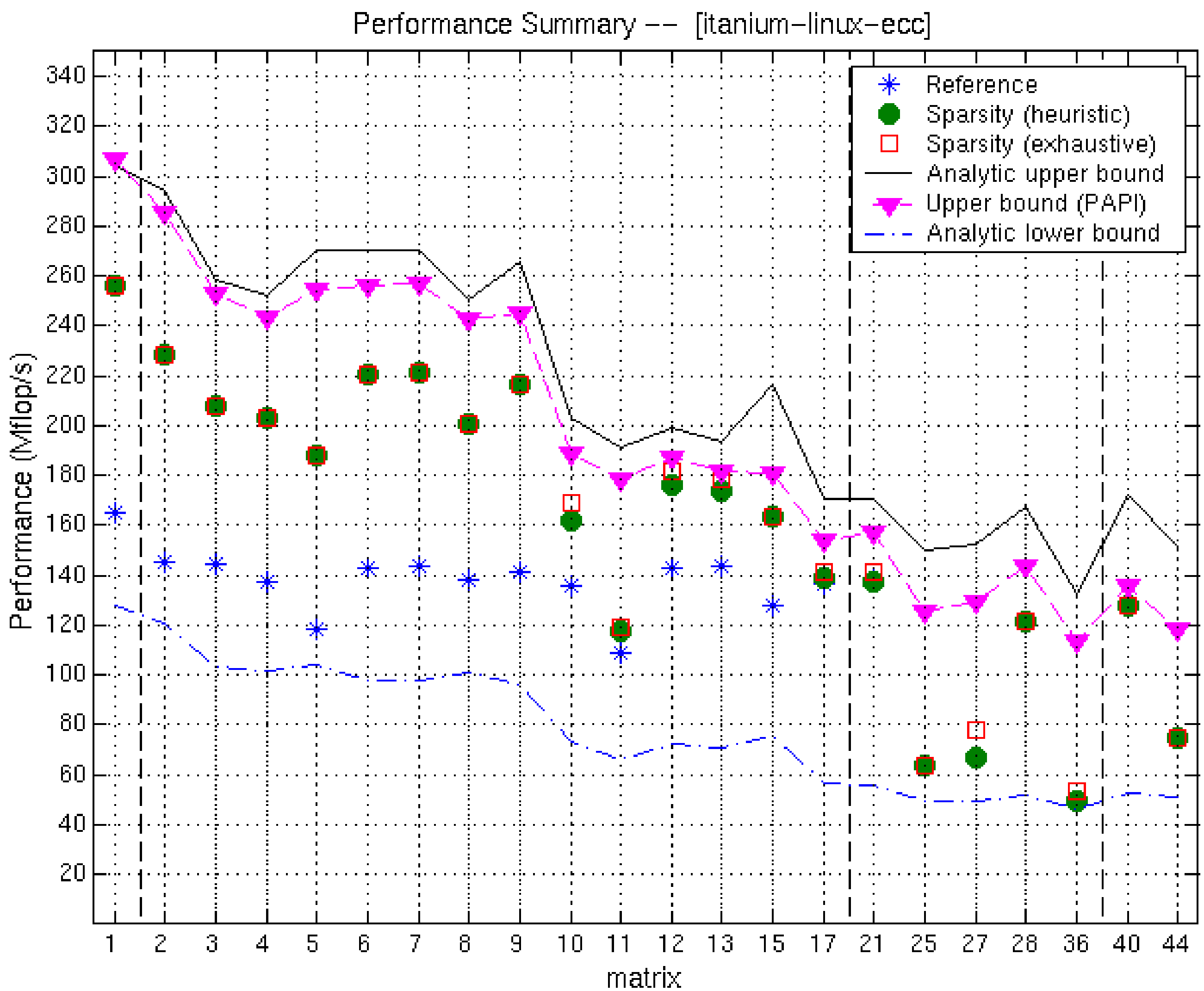
1500 Mflops

190 Mflops

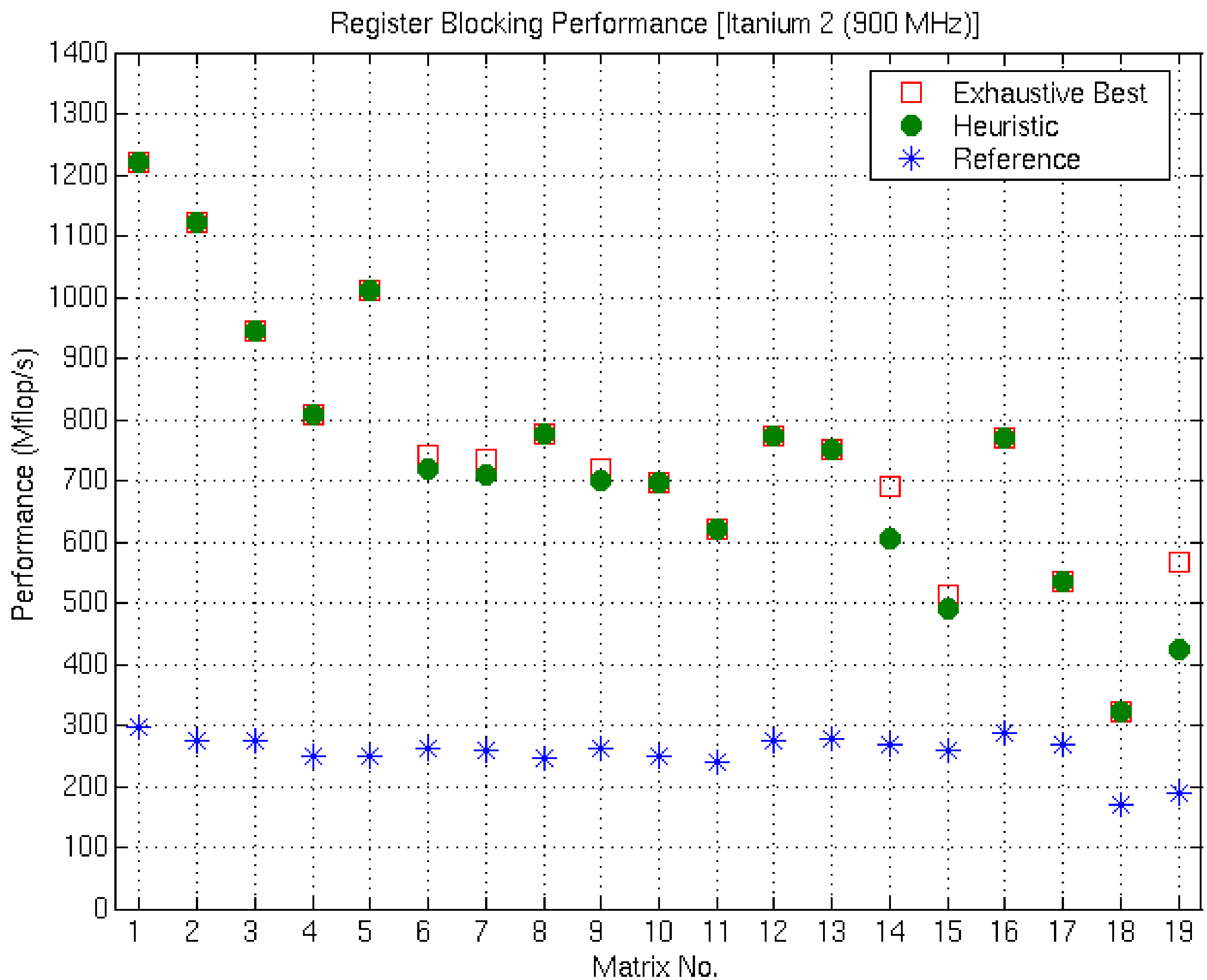
Pentium III SpMV speedups



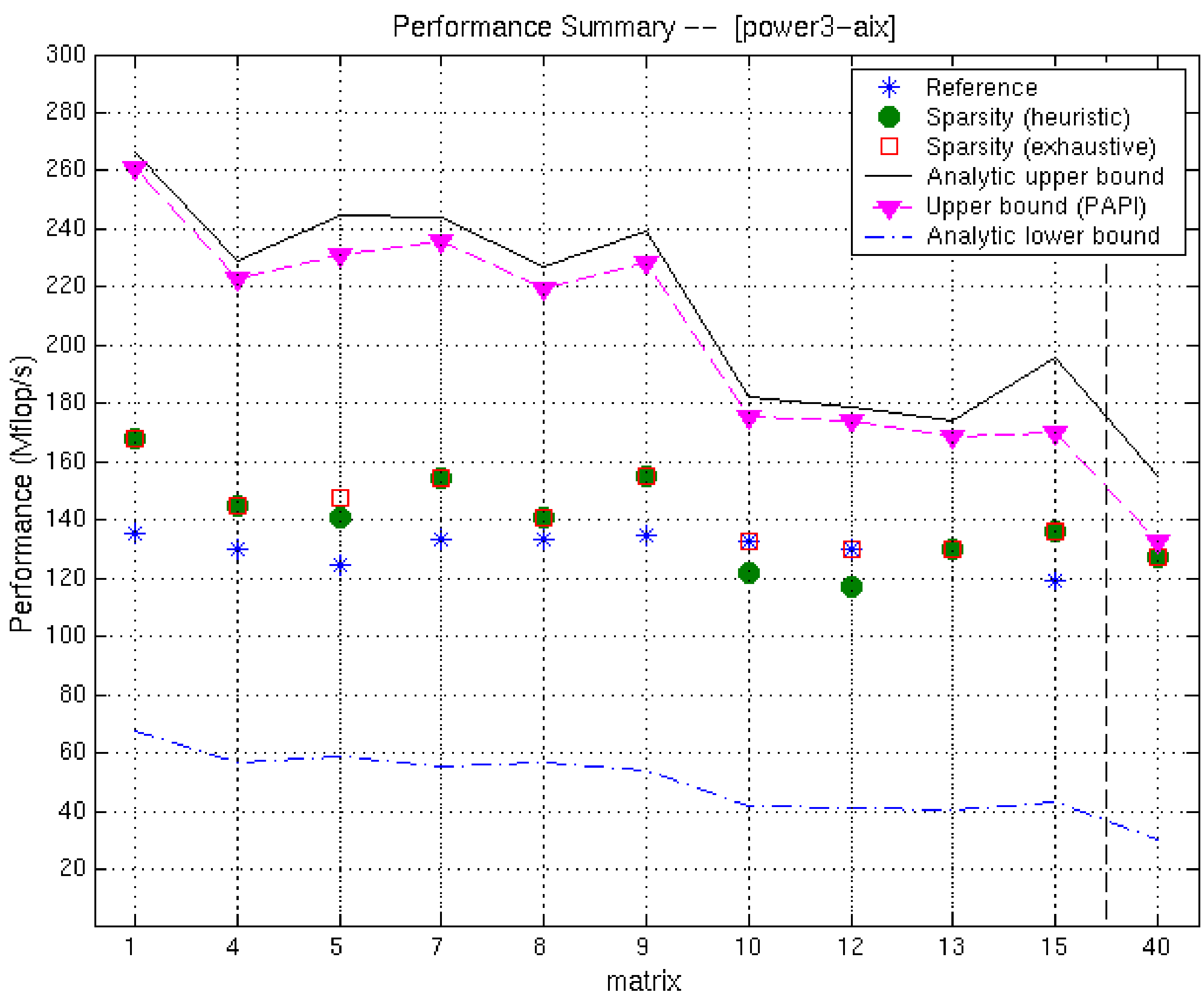
Itanium 1 SpMV Speedups



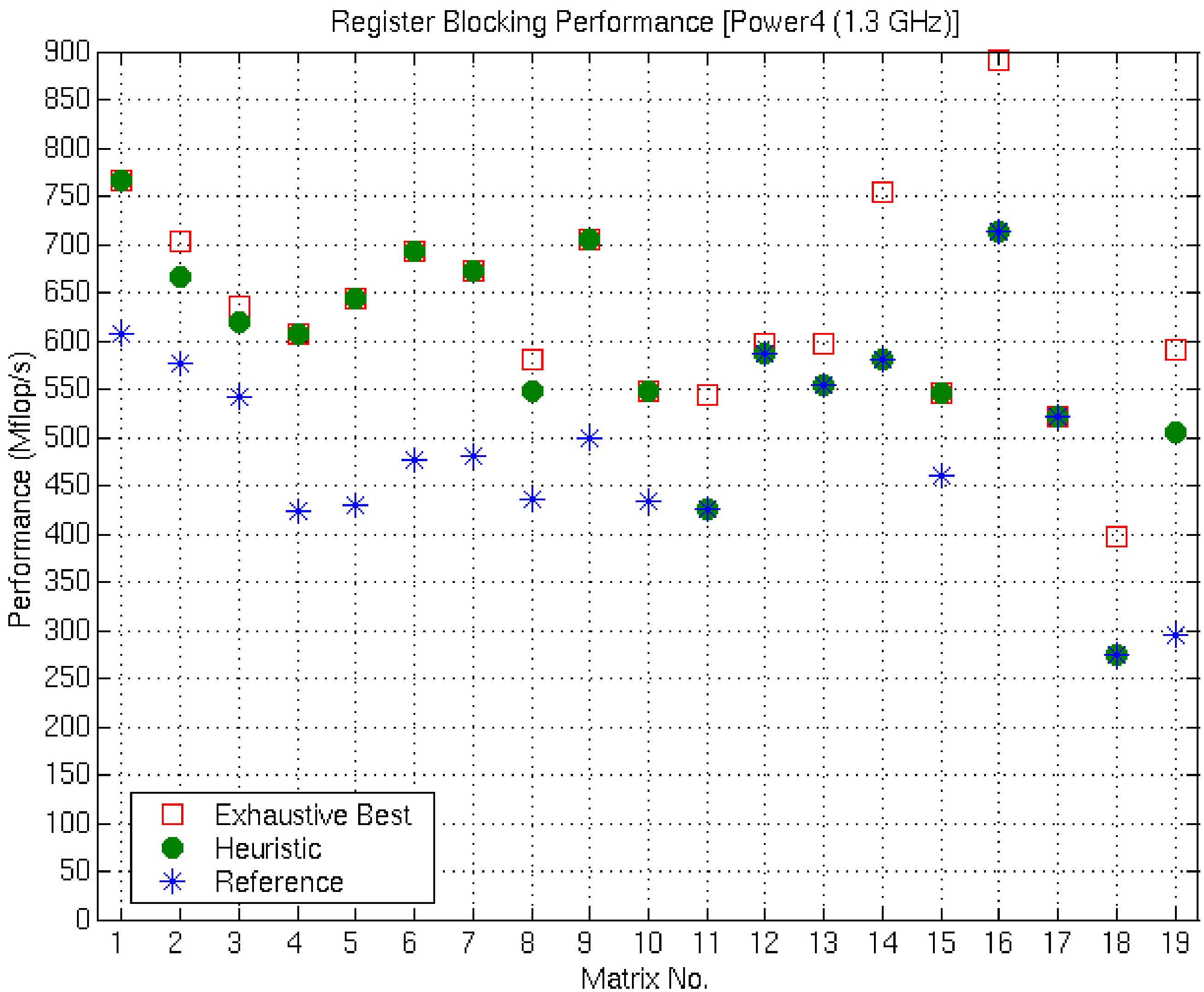
Itanium 2 SpMV Speedups



Power 3 SpMV Speedups

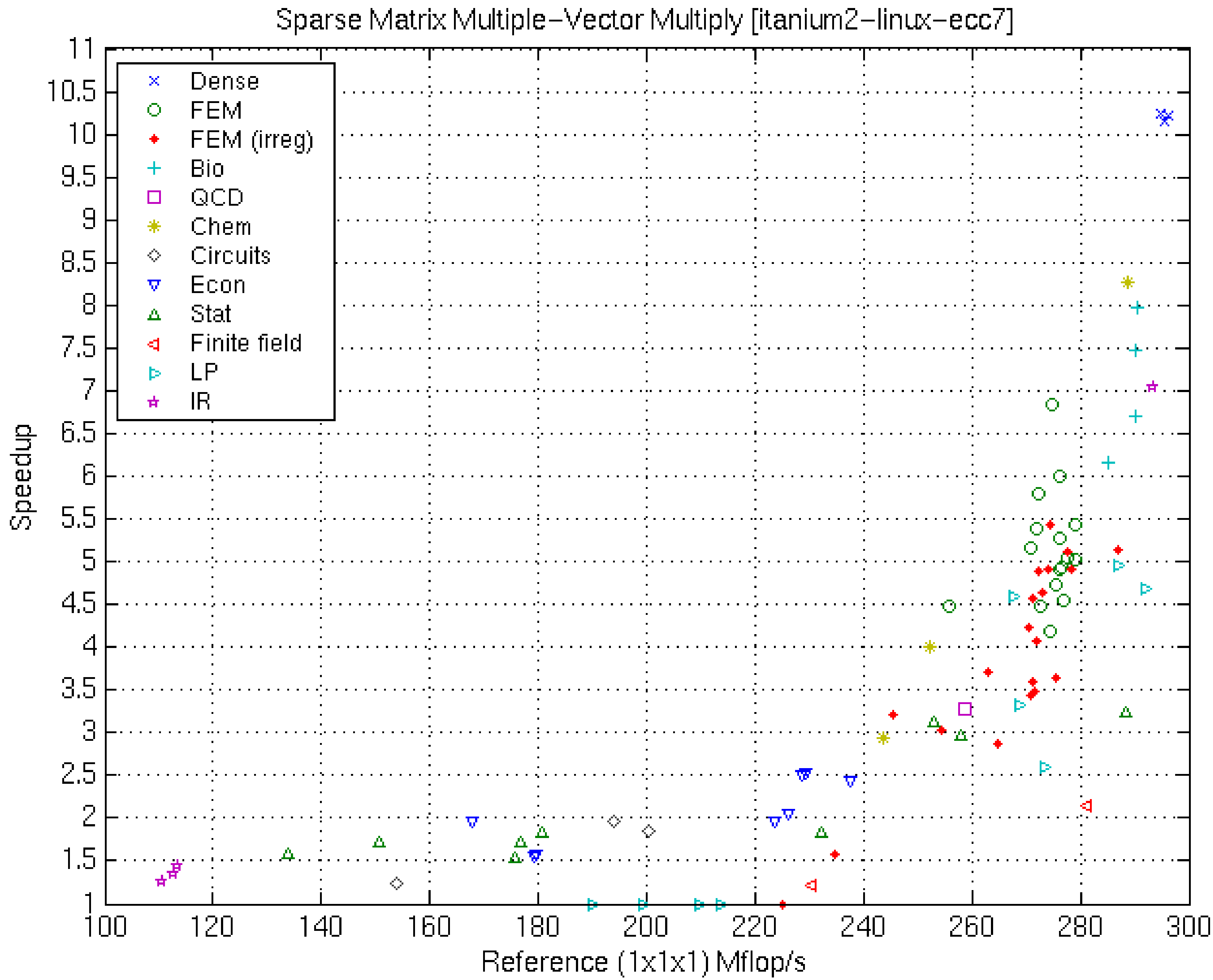


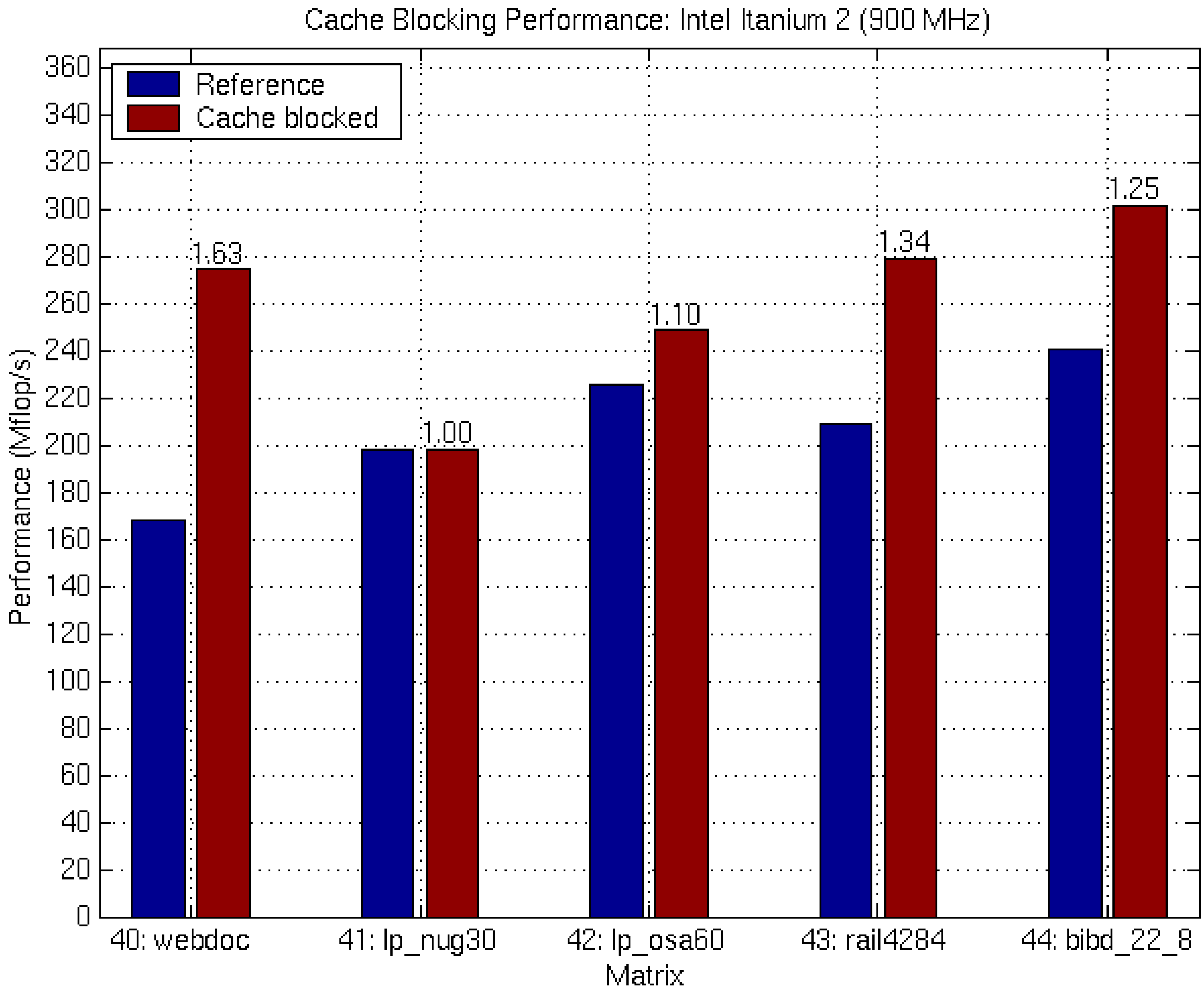
Power 4 SpMV Speedups



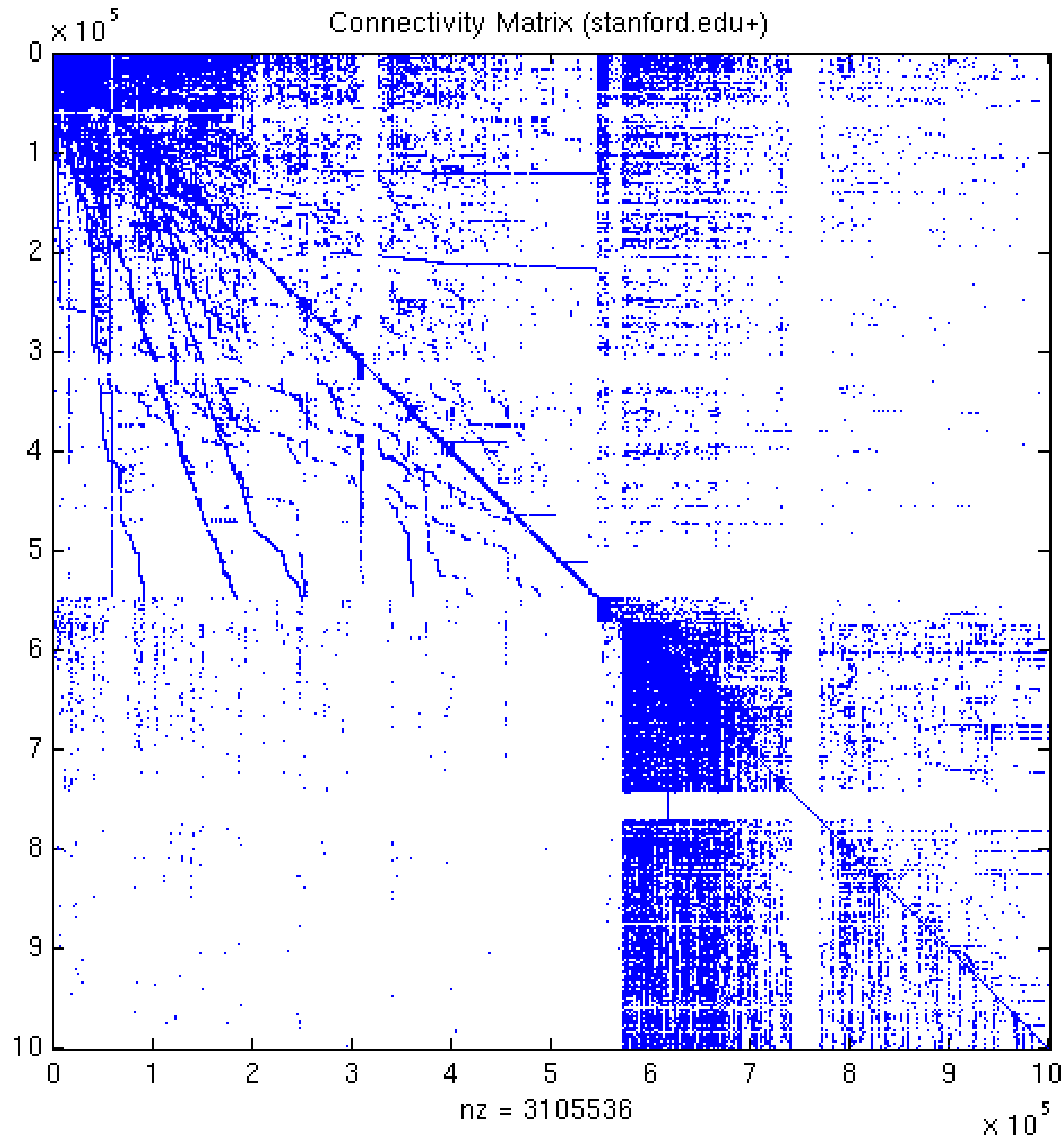
Exploiting Other Kinds of Structure

- Other optimizations for SpMV
  - Symmetry (up to 2x speedup)
  - Diagonals, bands (up to 2.2x)
  - Splitting for variable block structure (1.3x—1.7x)
  - Reordering to create dense structure + splitting (up to 2x)
  - Cache blocking (1.5—4x)
  - Multiple vectors (2—8x)
  - And combinations…
- Sparse triangular solve
  - Hybrid sparse/dense data structure (1.2—1.8x)
- Higher-level kernels
  - AA^Tx, A^TAx (1.2—4.2x)
  - RAR^T, A^kx, …

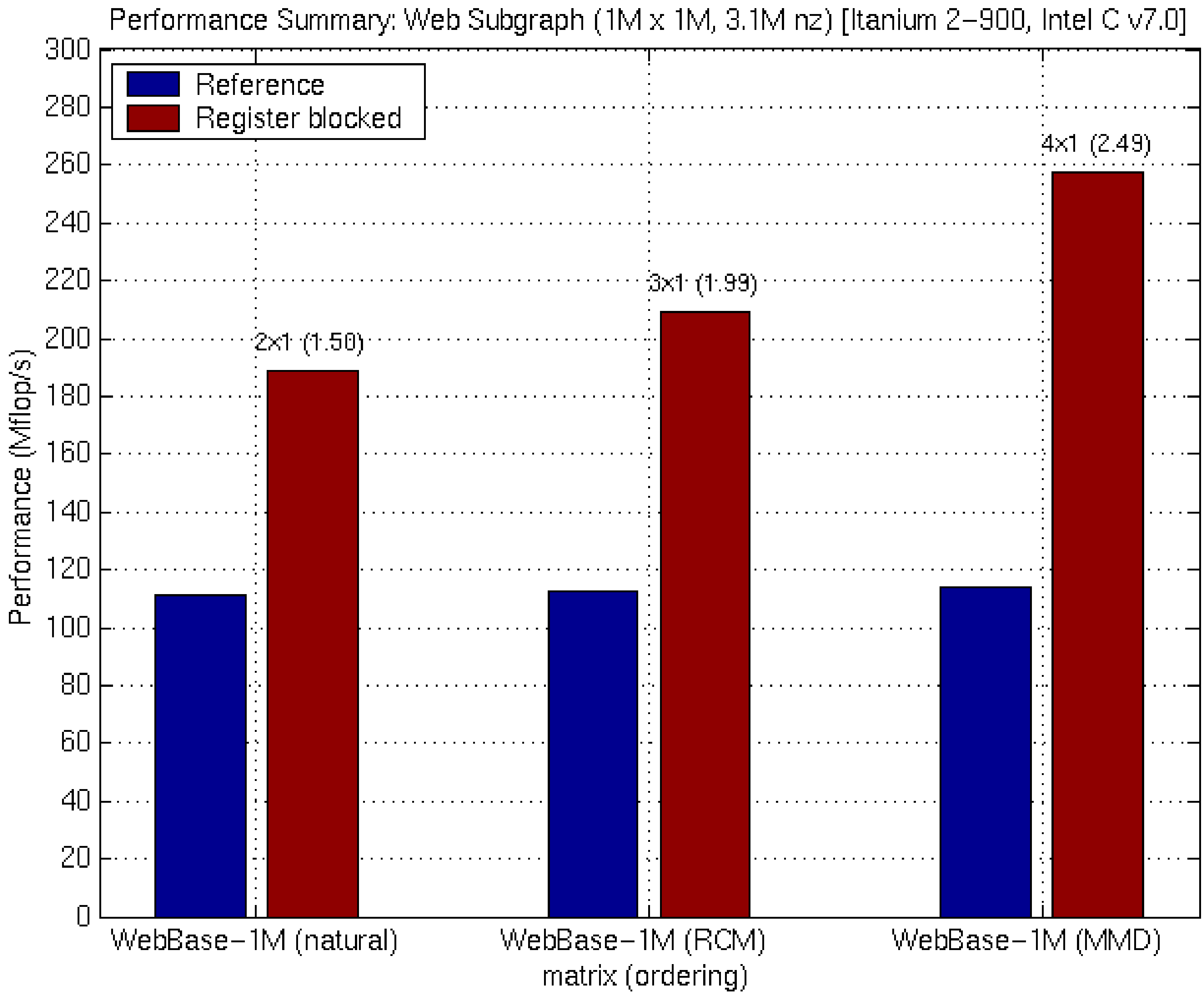

Sparse Matrix Multiple−Vector Multiply [itanium2−linux−ecc7]
Dense
FEM
FEM (irreg)
Bio
QCD
Chem
Circuits
Econ
Stat
Finite field
LP
IR
Speedup
Reference (1x1x1) Mflop/s
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10
10.5
11
100
120
140
160
180
200
220
240
260
280
300



Cache Blocking Performance: Intel Itanium 2 (900 MHz)
Reference
Cache blocked
Performance (Mflop/s)
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
1.63
1.00
1.10
1.34
1.25
40: webdoc
41: lp_nug30
42: lp_osa60
43: rail4284
44: bibd_22_8
Matrix

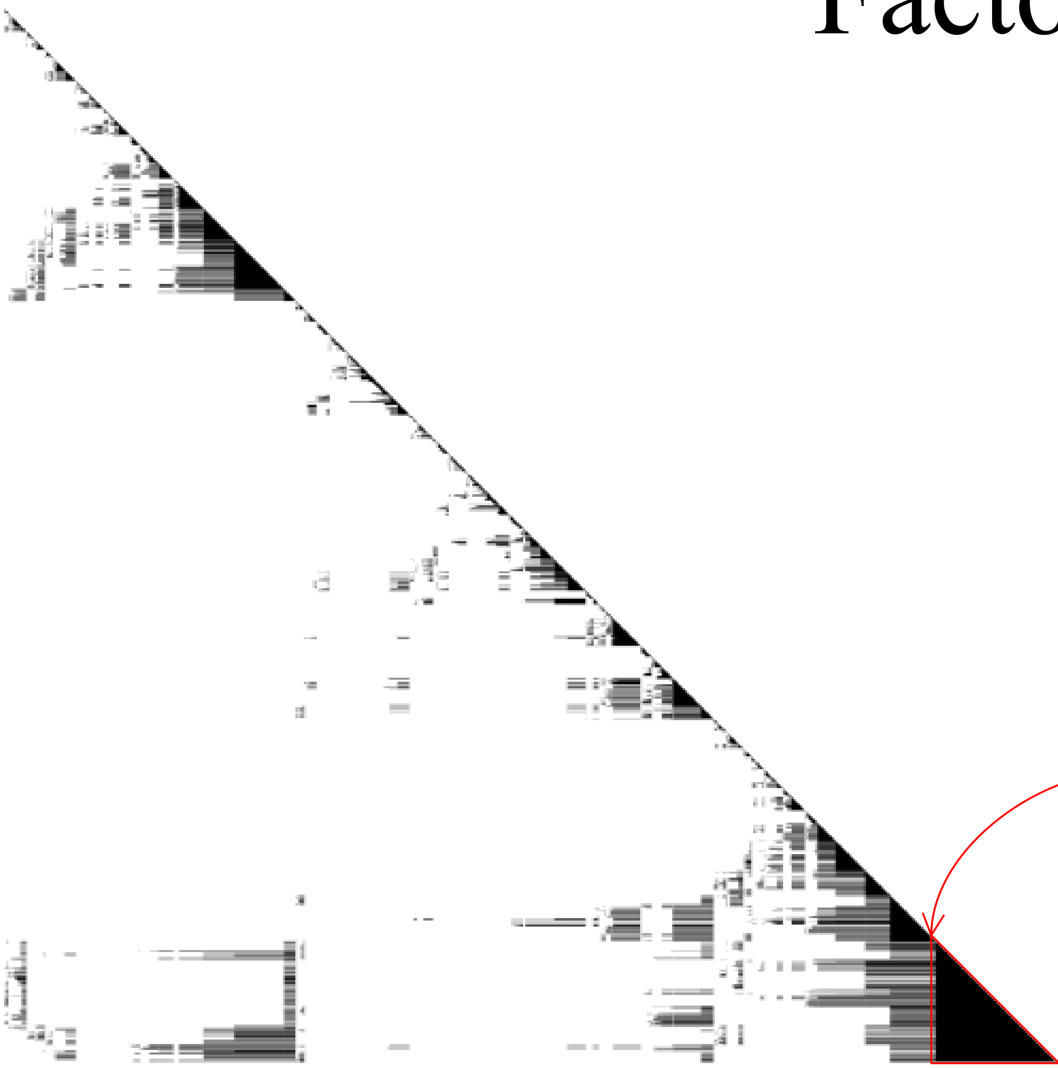


Connectivity Matrix (stanford.edu+)
× 10^5
0
1
2
3
4
5
6
7
8
9
10
nz = 3105536
× 10^5



Performance Summary: Web Subgraph (1M x 1M, 3.1M nz) [Itanium 2−900, Intel C v7.0]
Performance (Mflop/s)
Reference
Register blocked
2x1 (1.50)
3x1 (1.99)
4x1 (2.49)
WebBase−1M (natural)
WebBase−1M (RCM)
WebBase−1M (MMD)
matrix (ordering)
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300

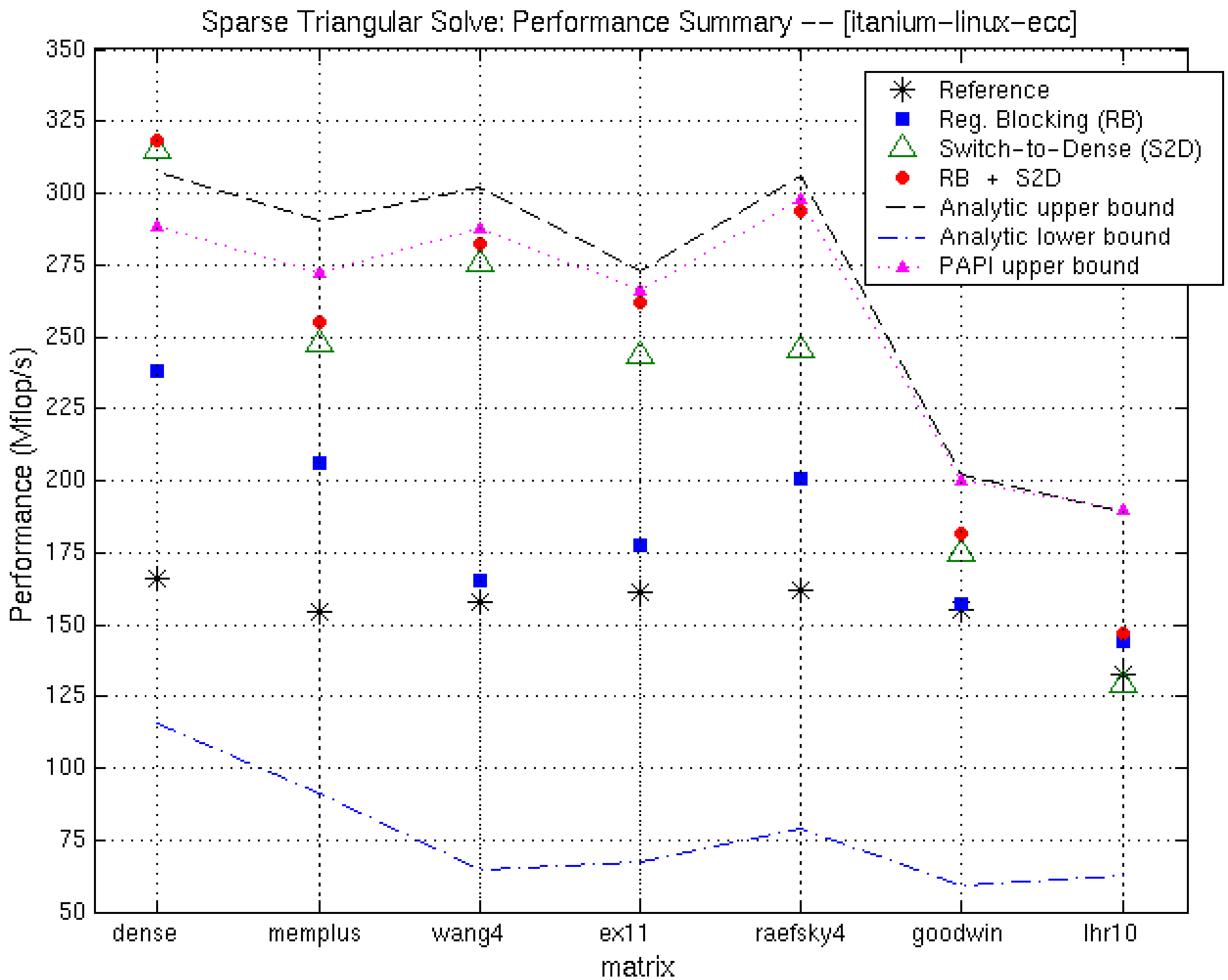
Example: Sparse Triangular Factor



Dense trailing triangle: dim=2268, 20% of total nz

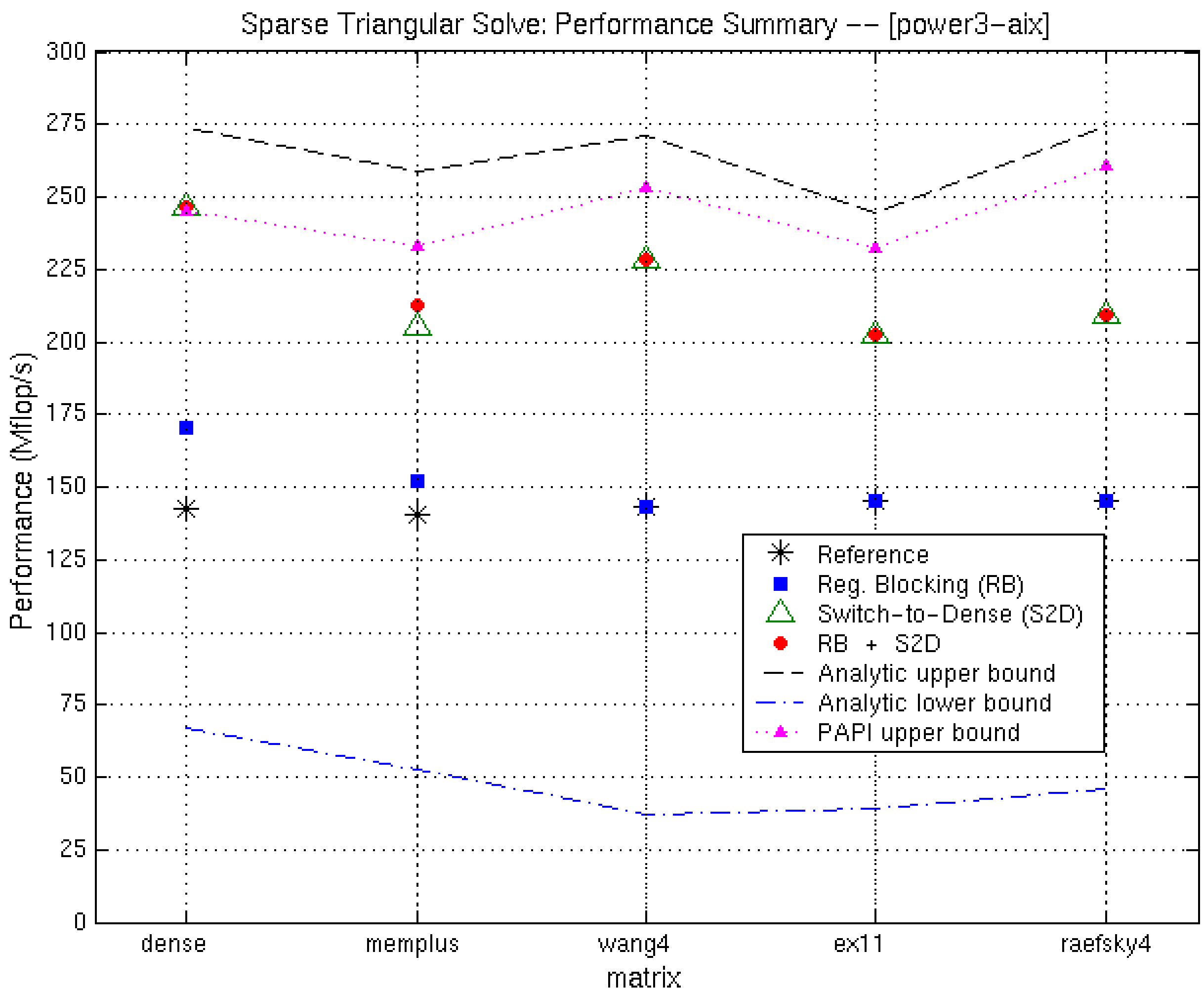
- Raefsky4 (structural problem) + SuperLU + colmmd
- N=19779, nnz=12.6 M

SpTS Performance: Itanium 1



(See POHLL '02 workshop paper, at ICS '02.)

SpTS Performance: Power3



Summary and Conclusions Automatic Performance Tuning

- Within 20% - 30% of peak for FE matrices
- Further improvements from new structure
  - Different data structures
  - New kernels
    - A symmetric (up to 2x)
    - A* multiple vectors (up to 8x)
    - A^T*A*x (up to 2x)
    - …
- Future –
  - use model to assess architectures
  - generalize to more kernels, embed in HLLs
- bebop.cs.berkeley.edu

Prometheus
A parallel distributed Multigrid for irregular meshes

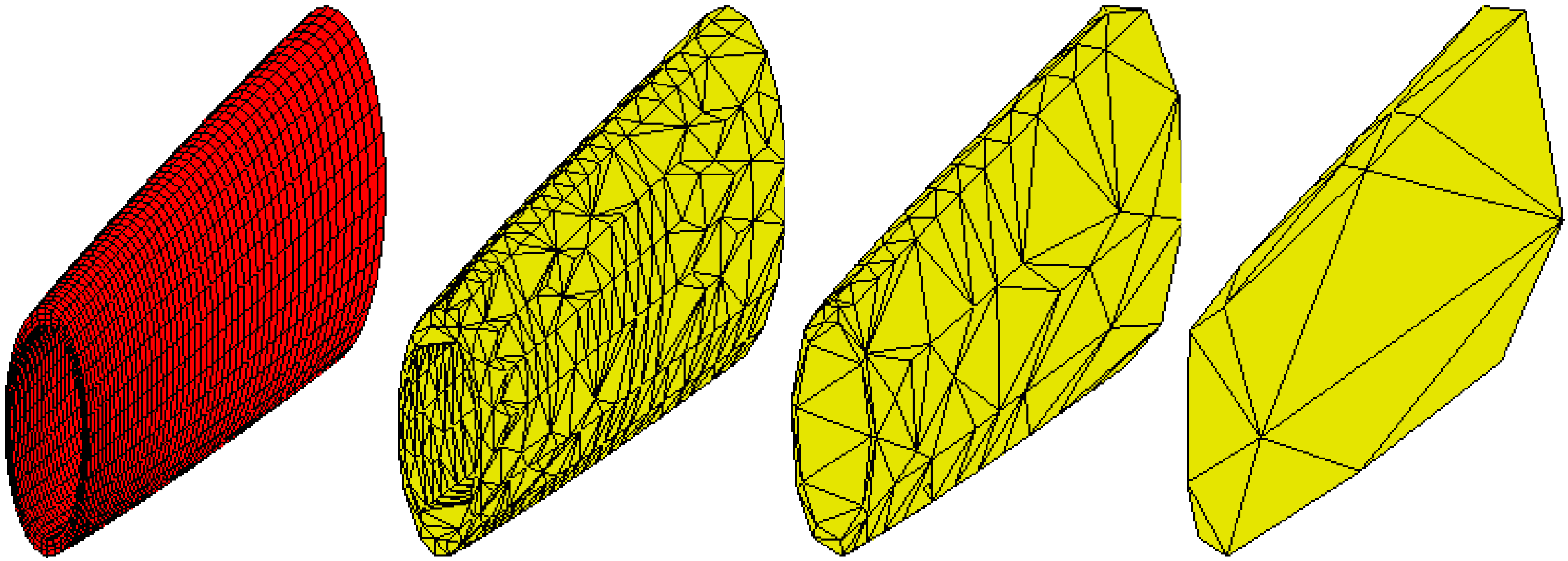
Mark Adams, Sandia NL
www.cs.berkeley.edu/~madams

(JD)

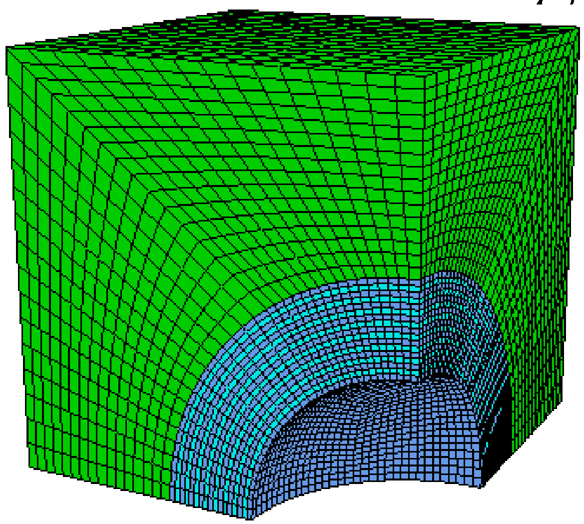
Multigrid on Irregular Meshes

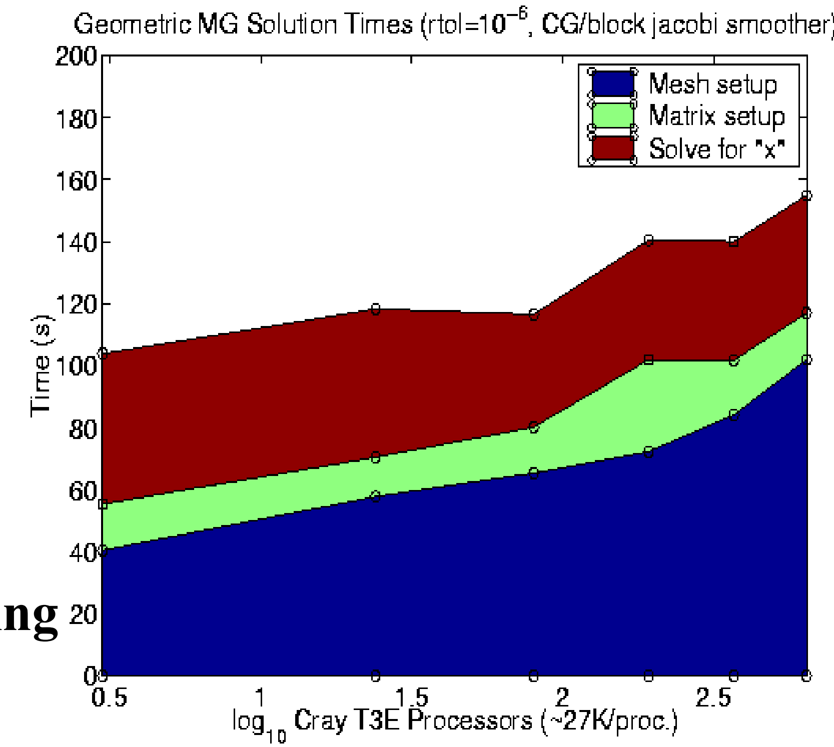
- Given fine grid matrix & mesh, coarsen
  - Solve recursively, using coarser grid to solve “low frequencies”
  - Goal – O(n) algorithm, linear speedup
- Geometric approach (Guillard, Chan, Smith)
  - Use Maximal Independent Sets, Delaunay meshing, to get coarse mesh from fine, using mesh coordinates
  - Use standard FE shape functions as restrictor
- Algebraic approach (Vanek, Brezina)
  - “Smoothed agglomeration”, no mesh coordinate
  - Aggregate strongly connected nodes
  - Use rigid body modes to construct prologation

Sample Coarse Grids



Prometheus – Parallel Multigrid Solver for Irregular FE Problems

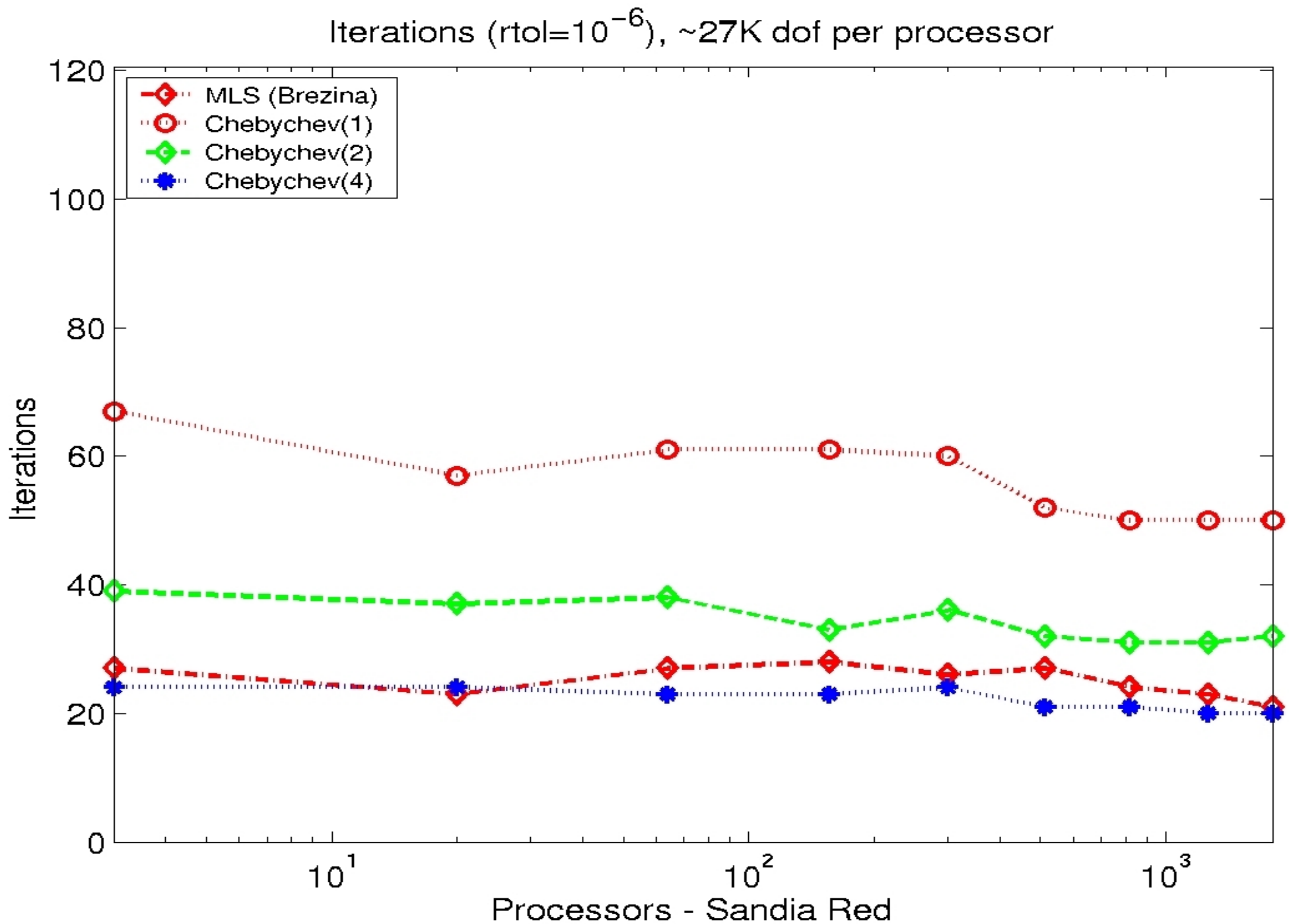




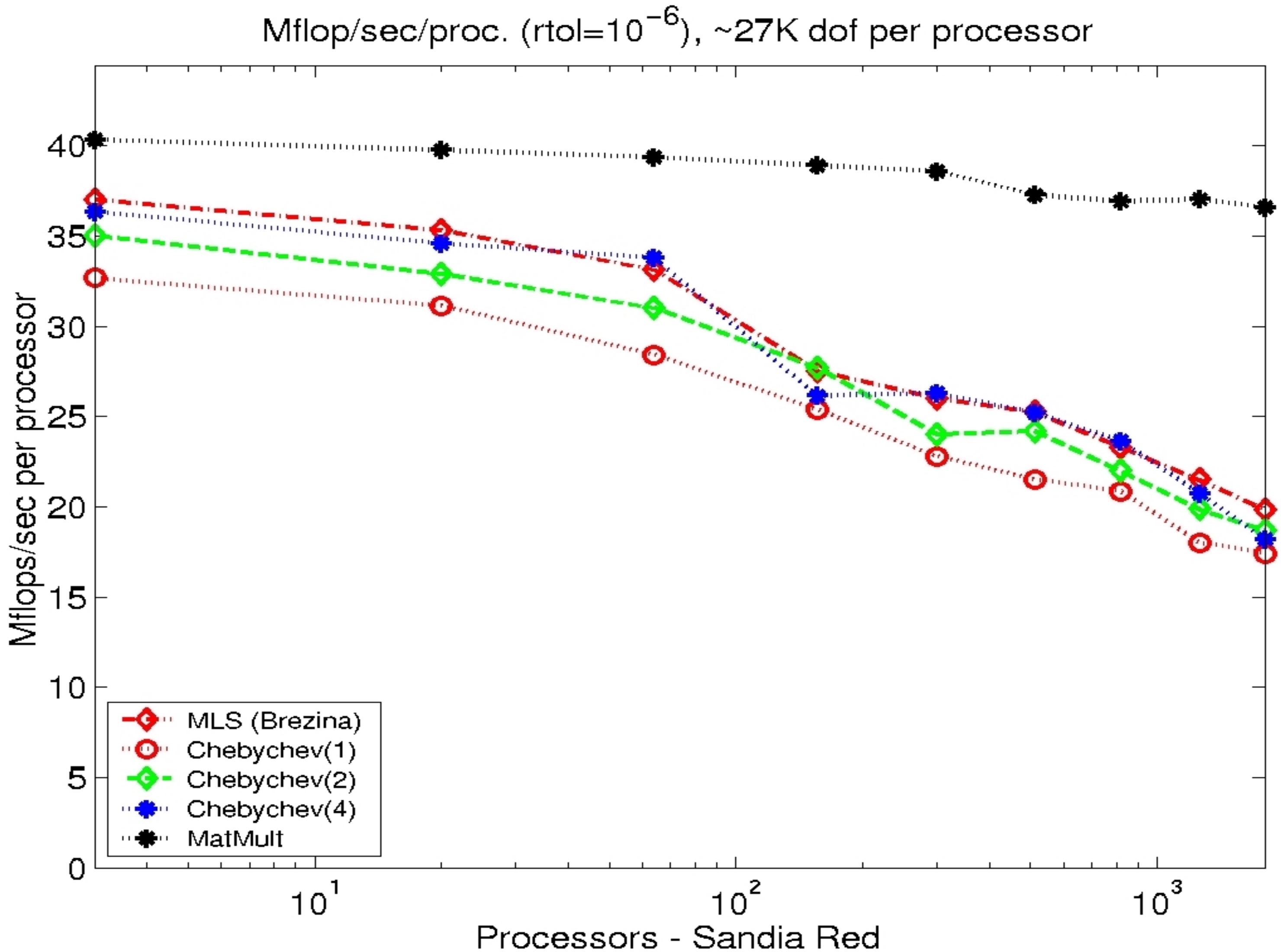
- **Stiff sphere w/ 17 steel and rubber layers, embedded in rubber cube; compute crushing**
- **80K – 56M dof**
- **Up to 640 Cray T3E processors**
- **50% scaled parallel efficiency**
- **76M dof solved in 70 seconds on 1920 processor ASCI Red (SC ’01)**
- **Prize for Best Industrial Appl in Mannheim SuParCup 99**

www.cs.berkeley.edu/~madams

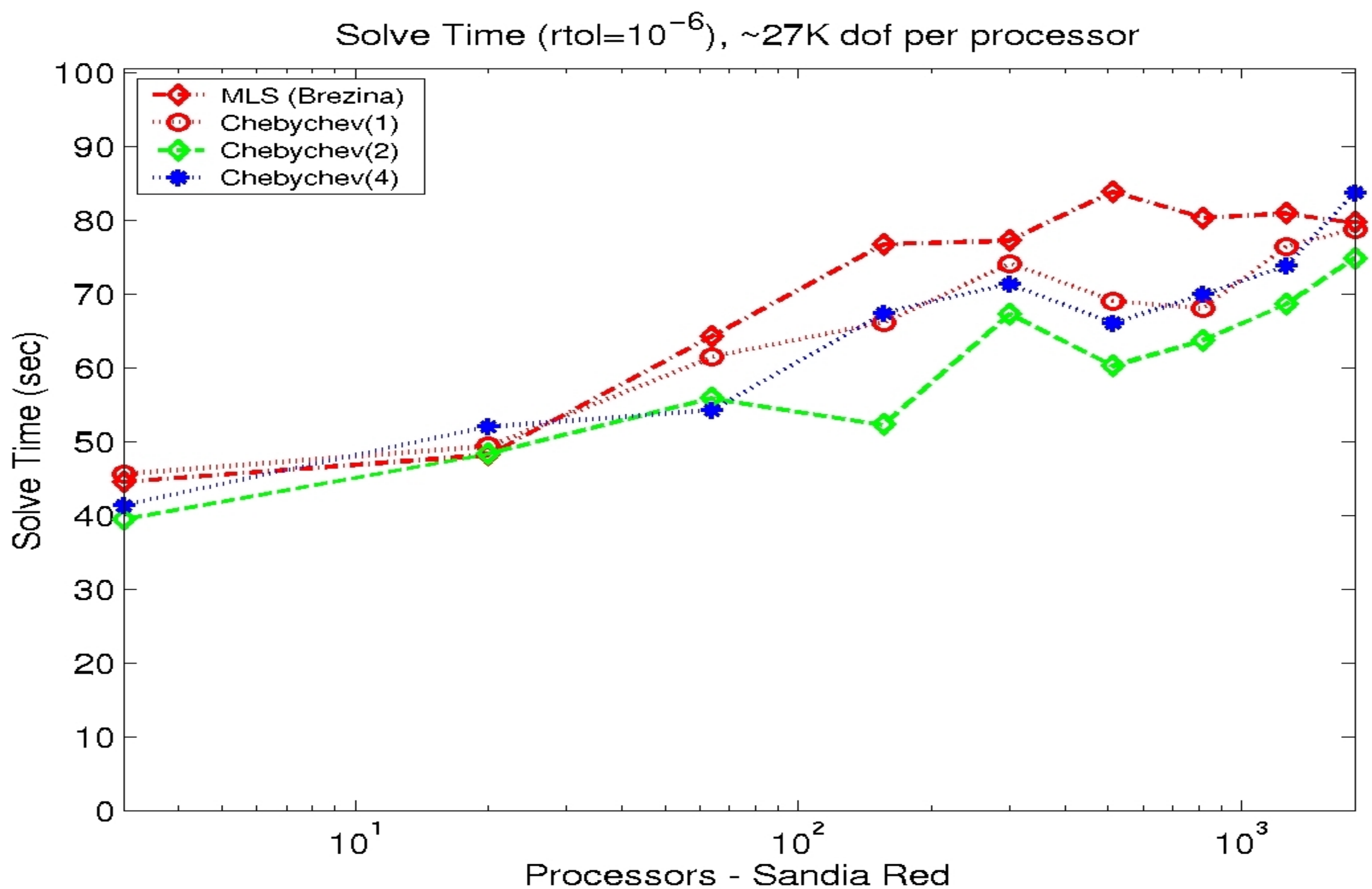
#MG iterations on 1800 PE ASCI Red



Performance on 1800 PE ASCI Red



Total Solve Time on 1800 PE ASCI Red



Conclusions

- Discussed scalability for linear algebra
  - Dense
  - Sparse Direct
  - Sparse Iterative (Multigrid)
- Many algorithms scale well (on model problems)
  - Many performance models available
  - Better algorithms under development
  - Automatic performance tuning helps
- Expect latency to be issue for sparse problems on very large machines